N° d'ordre : 2330

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **David GOUDIN**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Mise en œuvre d'une Bibliothèque d'Outils pour la Résolution Parallèle Hautes Performances par Méthode Directe de Grands Systèmes Linéaires Creux et application à un Code de Mécanique des Structures.

Soutenue le : 30 novembre 2000

Après avis des rapporteurs :

Patrick Amestoy	Maître de Conférence
Daniel Bouche	Directeur de Recherche

Devant la commission d'examen composée de :

Patrick Amestoy	Maître de Conférence	Rapporteur
Daniel Bouche	Directeur de Recherche ...	Rapporteur
Richard Castanet ...	Professeur	Président
Jean Jacques Pesqué	Ingénieur E.N.S.E.E.I.H.T.	Examinateur
Jean Roman	Professeur	Directeur de thèse
Yousef Saad	Professeur	Examinateur

– 2000 –

Remerciements

Je tiens vivement à remercier ici :

Patrick Amestoy, maître de conférences à l'IRIT-Toulouse, pour avoir bien voulu rapporter sur ce travail. Ses nombreuses remarques m'ont permis de compléter utilement ce document et son aide a été précieuse ;

Daniel Bouche, directeur de recherche associé à l'ENS Cachan et adjoint au chef du département de Physique Théorique et Appliquée du Centre de BIII du CEA pour l'honneur qu'il m'a fait en acceptant de rapporter sur ce travail ;

Richard Castanet, professeur et directeur du LaBRI, pour avoir accepté de présider ce jury ;

Yousef Saad, professeur à l'université du Minnesota, pour l'immense honneur qu'il m'a fait en acceptant d'être présent dans mon jury malgré de nombreuses sollicitations ;

Jean-Jacques Pesqué, adjoint au chef du département Informatique Scientifique du CEA/CESTA, pour avoir contribué à l'aboutissement de cette thèse, pour son dynamisme, sa gentillesse et sa patience.

Un grand merci à Jean Roman, professeur à l'ENSEIRB-Bordeaux, pour sa confiance, sa gentillesse et son sérieux tout au long de ces trois années d'encadrement de ma thèse. Sans ses compétences et sa rigueur dans le travail, cette bibliothèque logicielle et ce travail de thèse n'auraient jamais pu aboutir.

Tout ce travail n'aurait pu aboutir sans l'aide et la présence des personnes que je tiens à remercier ici.

En premier lieu, je veux remercier l'équipe entière de la bibliothèque EMILIO : François (grâce à qui le brevetage logiciel ne passera pas), Pascal (Sedan et Dudule 5 - PSG 1) et Pierre (allume les strobos et la techno) pour leur amitié et leur aide lors des bons et (surtout) des mauvais moments. De même je remercie l'équipe ALiENor du LaBRI de m'avoir accueilli et supporté lors de ces trois années.

Je remercie l'ensemble des actuels et anciens membres de la CVT pour l'ambiance et le soutien qu'ils m'ont apporté lors de ces trois années : Asier (le compagnon basque de nombreux sandwichs mangés le midi), Pierre (qui a compris tous les exposés à Aussois, d'après ce qu'il dit), Pascal (le petit nouveau qui monte), Macha, Philippe, Gérald, Antoine, Isabelle, Angel, Olivier, Richard et les autres....

Merci à Pierre André pour son soutien et ses conseils, ainsi que les autres doctorants : Valère, Davy, Nicolas, Akka, Yvan, Yon, Sami, ... pour les pauses cafés et discussions très souvent distrayantes.

Merci aux membres de l'équipe Analyse Numérique du service informatique du CEA/CESTA qui vont m'accueillir pendant l'année à venir : Michel, Murielle, Bernard, Mr Magne, etc...

Un merci spécial au club de Basket de Gardonne, non seulement l'équipe première : Nicolas, Bertrand, Thomas, Sébastien, Yoan, Benoît, Marin, Quentin, Stéphane, Dominique pour les rires (beaucoup) et les sueurs (moins) partagés ensemble surtout lors des stages de présaison à Saint Lary ; mais aussi notre président si dévoué : Gérard, notre entraîneur : Christophe, l'ancien entraîneur : Christian et l'ensemble des équipes du club. Grâce à eux, j'ai pu oublier les moments pénibles le temps d'entraînements, de matchs et de trajets en bus.

Je remercie toutes les personnes que je ne vois pas assez souvent pour des raisons diverses et variées mais que je suis très heureux de connaître : Frédéric le spécialiste des Fêtes de Bayonne, Philippe le buveur de mazout, Eric le footballeur, Vincent dit “Petit Tour”, Stéphane le Bidochon, Cyril et Richard les Toulousains, Richard et Marc de Ambris, Jean Philippe “l'Ange Gardien”.

A mes amis de longue date (que c'est loin l'IUT !), Karine et Eric avec qui j'ai partagé tant de mails et de discussions de fin de soirée, je tiens à dire un grand merci et à bientôt pour une grande fête. Je veux aussi remercier Jean Claude, mon coéquipier de pétanque, pour son aide inestimable et son amitié infaillible depuis de très longues années.

Un merci immense à ma mère Colette qui a consenti tant de sacrifices pour que mon cursus se passe le mieux possible, son ami Claude pour sa gentillesse naturelle et son aide.

A Marie, qui m'accompagne depuis 6 ans, je veux dire un merci plein de tendresse et de gratitude car sans elle rien n'aurait été possible.

A mes grands parents maternels Odette et Pierre qui ne sont malheureusement plus de ce monde, je n'ai pas de mot pour exprimer toute ma reconnaissance, seulement qu'ils me manquent cruellement ...

Mise en œuvre d'une Bibliothèque d'Outils pour la Résolution Parallèle Hautes Performances par Méthode Directe de Grands Systèmes Linéaires Creux et application à un Code de Mécanique des Structures

Résumé : Cette thèse, qui concernait initialement la parallélisation du code OSSAU du CEA/CESTA - code industriel vectorisé de mécanique des structures, non linéaire en temps et en deux ou trois dimensions d'espace - a débouché sur la conception et l'implémentation d'une chaîne complète de traitement parallèle hautes performances pour l'assemblage et la résolution par méthodes directes de grands systèmes linéaires creux. L'objectif final est une validation en vraie grandeur sur le code OSSAU pour des problèmes 3D ayant plusieurs millions d'inconnues.

Mots clés : calcul hautes performances, matrice creuse, solveurs directs parallèles, éléments finis, assemblage parallèle, mécanique des structures.

Discipline : Informatique

LaBRI,
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

Implementation of a Software Processing Chain for the Parallel High Performance Resolution of Sparse Linear Systems by Direct Methods and application to a Structural Mechanics Code

Abstract : This thesis initially concerned the parallelization of the OSSAU software code from CEA/CESTA. The application domain of this software is the vectorized structural mechanics ; the code itself is non linear in time and in two or three dimensions. This investigation leads to the conception and the implementation of a parallel high performance software processing chain for the assembly and the resolution of sparse linear systems by direct methods. The final objective is a validation of the OSSAU code for three dimension problems with several millions of unknowns.

Keywords : high performance computation, sparse matrix, parallel direct solvers, finite elements, parallel assembly, structural mechanics.

Discipline : Computer Science

LaBRI,
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

Table des matières

Remerciements **i**

Introduction générale **1**

I Cadre de l'étude 5

1 Le problème mathématique **7**

1.1 Rappel de la méthode des Eléments Finis 7
1.1.1 Elasticité linéaire 7
1.1.1.1 Assemblage 10
1.1.1.2 Calcul des matrices de rigidité élémentaires K_e . . . 10
1.1.2 Problèmes non linéaires 12
1.1.2.1 Méthodes de résolution 13
1.1.2.2 Méthode du point fixe ("Rigidité sécante") 13
1.1.2.3 Méthode de Newton ("Rigidité tangente") 13
1.1.2.4 Méthodes incrémentales 14
1.1.2.5 Cas de la plasticité 16
1.1.2.6 Cas des grands déplacements : non linéarité géométrique 18
1.1.3 Choix de la méthode de résolution du système linéaire 18
1.1.3.1 Propriétés de ces systèmes linéaires 19
1.1.3.2 Etude succincte du conditionnement 20
1.1.3.3 Méthodes itératives 21

2 Cadre général **23**

2.1 Introduction 23
2.2 La renumérotation 23
2.3 La factorisation symbolique 24
2.4 Repartitionnement et distribution des blocs de la matrice 26
2.5 Résolution numérique 27

II Description de la chaîne logicielle EMILIO 29

3 Introduction et étapes liées à la résolution 31

3.1 Enchaînement pour le prétraitement séquentiel des données 32

3.1.1 Renumérotation et Factorisation Symbolique par Blocs 35

3.1.2 Repartitionnement et distribution des blocs 37

3.2 Enchaînement pour le traitement parallèle 39

3.2.1 Le solveur parallèle 41

4 Distribution des éléments 47

4.1 Introduction 47

4.2 Algorithmique et mise en œuvre 47

4.3 Validation expérimentale de la distribution 53

4.3.1 COLOGB75 avec 64 processeurs 54

4.3.2 CEL3DM avec 64 processeurs 55

4.3.3 COUPOLE3000T avec 64 processeurs 56

4.4 Analyse des résultats 57

5 Assemblage parallèle 59

5.1 Introduction 59

5.2 Algorithmique et mise en œuvre 59

5.3 Validation expérimentale de l'assemblage 64

III Validation Expérimentale 69

6 Gestion de la mémoire 71

6.1 Gestion de la mémoire dans le projet EMILIO 71

6.2 Tailles mémoire nécessaires pour le code OSSAU original 72

7 Le cas de Cologne 75

7.1 Description du Problème 75

7.1.1 Test de Cologne 75

7.1.2 Tests de Cologne allongés en X 78

7.2 Validation expérimentale du cas de Cologne 80

7.2.1 Description des maillages 80

7.2.2 Résultats 80

7.2.2.1 Distribution des éléments 80

7.2.2.2 Assemblage et résolution 81

8 Composite Tissé 3D **87**
8.1 Description du Problème 87
8.2 Validation expérimentale des cas CEL3DM et CEL3DMB20 91
8.2.1 Description des maillages 91
8.2.2 Résultats 91
8.2.2.1 Distribution des éléments 91
8.2.2.2 Assemblage et résolution 92

9 Coupole sous chargement "linéique" **95**
9.1 Description du Problème 95
9.2 Validation expérimentale du cas Coupole 97
9.2.1 Description des maillages 97
9.2.2 Résultats 97
9.2.2.1 Distribution des éléments 97
9.2.2.2 Assemblage et résolution 98

Conclusion générale et perspectives **101**

Annexe 103

A Conditions expérimentales **105**

B Temps d'exécution des modules séquentiels **107**

C Temps de factorisation parallèle pour une distribution bloc 1D **109**

D Diagrammes de Gantt et de communication **111**

E Temps pour la descente-remontée parallèle 1D **115**

F Temps de factorisation parallèle pour une distribution 2D/1D **117**

G Comparaison de performances avec le logiciel PSPASES **119**

H Liste des publications de l'auteur **121**

Liste des tableaux

5.1 Temps d'assemblage de pseudo matrices élémentaires. 64
6.1 Tailles mémoires en octets pour le code OSSAU. 73
7.1 Description numérique des cas du test de Cologne. 80
7.2 Temps de distribution des éléments pour les cas du test de Cologne. . 81
7.3 Temps des étapes parallèles pour une itération de COLOGB00. 82
7.4 Temps des étapes parallèles pour une itération de COLOGB15. 83
7.5 Temps des étapes parallèles pour une itération de COLOGB30. 83
7.6 Temps des étapes parallèles pour une itération de COLOGB75. 84
8.1 Description du maillage du cas test CEL3DM. 91
8.2 Description du maillage du cas test CEL3DMB20. 91
8.3 Temps de distribution des éléments pour le cas test CEL3DM. 92
8.4 Temps de distribution des éléments pour le cas test CEL3DMB20. . . 92
8.5 Temps des étapes parallèles pour une itération de CEL3DM. 92
8.6 Temps des étapes parallèles pour une itération de CEL3DMB20. . . . 93
9.1 Description des maillages des cas tests Coupole. 97
9.2 Temps de distribution des éléments pour les cas tests Coupole. 97
9.3 Temps des étapes parallèles pour une itération de COUPOLE20T. . . 98
9.4 Temps des étapes parallèles pour une itération de COUPOLE450T. . 98
9.5 Temps des étapes parallèles pour une itération de COUPOLE1500T. . 98
9.6 Temps des étapes parallèles pour une itération de COUPOLE2000T. . 99
A.1 Description des matrices tests. 106
B.1 Temps d'exécution des étapes de partitionnement/distribution et factorisation symbolique en 1D. 108
C.1 Temps d'exécution de la factorisation parallèle en 1D pour des cas réguliers. 109
C.2 Temps d'exécution de la factorisation parallèle en 1D pour des cas irréguliers. 110
E.1 Temps d'exécution de la résolution parallèle en 1D pour des cas réguliers. 115

E.2 Temps d'exécution de la résolution parallèle en 1D pour des cas irréguliers. 116
F.1 Temps d'exécution de la factorisation parallèle en 2D pour des cas réguliers. 117
F.2 Temps d'exécution de la factorisation parallèle en 2D pour des cas irréguliers. 118
G.1 Description des cas tests de comparaison. 120
G.2 Comparaison des performances avec PSPASES. 120

Liste des figures

1.1 Un hexaèdre à interpolation linéaire et Elément de référence 11
1.2 Le cas de la plasticité 16
1.3 La courbe déplacement 20
2.1 Une matrice factorisée structurée par bloc et son arbre d'élimination. 25
3.1 L'enchaînement des différentes étapes 32
3.2 Les étapes du prétraitement séquentiel 34
3.3 Sans Halo. 36
3.4 Avec Halo. 36
3.5 Un exemple de matrice factorisée structurée par bloc colonne. . . . 37
3.6 Les étapes parallèles du projet EMILIO. 40
3.7 Un exemple de matrice factorisée structurée par bloc. 42
3.8 Algorithme de la factorisation parallèle pour une distribution 1D. . 43
3.9 Modifications apportées à l'algorithme de factorisation 1D. 44
3.10 Algorithme de la factorisation parallèle pour une distribution 2D. . 45
4.1 Un exemple de dissections sur une grille 5×5. 49
4.2 L'arbre d'élimination correspondant à la grille 5×5 50
4.3 Le graphe des éléments de COLOGB15. 51
4.4 Le graphe des nœuds pour COLOGB15. 52
4.5 Dissections emboîtées (4 niveaux de récursion) obtenues par SCOTCH sur COLOGB15. 52
5.1 Degrés de Liberté numérotés continuement pour un même nœud. . . 60
5.2 Algorithme de l'assemblage parallèle pour une distribution 1D. . . . 61
5.3 Diagramme de Gantt pour COLOGB75 (0.01 s). 66
5.4 Diagramme de Gantt pour COLOGB75 (0.05 s). 66
5.5 Diagramme de Gantt pour COLOGB75 (0.1 s). 67
6.1 L'utilisation de la mémoire 72
D.1 CUBE39 : diagramme de Gantt et de communications. 112
D.2 BMW3_2 : diagramme de Gantt et de communications. 112
D.3 CRANKSG1 : diagramme de Gantt et de communications. 113

Introduction générale

La simulation numérique intensive est incontestablement un domaine où les techniques de calcul hautes performances sont incontournables. L'utilisation de calculateurs parallèles ayant un (très) grand nombre de processeurs repousse la limite de ce que l'on savait faire jusqu'à maintenent en termes de qualité de simulation. On peut donc traiter plus rapidement une physique plus complexe (problèmes couplés, non linéarité difficiles, ...) sur des domaines plus finement maillés. Seule une parallélisation peut apporter les espaces mémoires et la puissance C.P.U. nécessaires pour traiter ce type de simulation avec des temps de résolution raisonnables.

Ces applications de grandes tailles sont souvent de nature irrégulière (par exemple à cause de la manipulation de très grandes matrices creuses), et organisées comme des enchaînements d'étapes algorithmiques qu'il s'agit de paralléliser de manière très efficace pour obtenir de bonnes performances globales. La manière de gérer l'enchaînement lui-même sur le calculateur parallèle peut être aussi une source de difficultés.

Ces étapes algorithmiques comportent souvent des noyaux d'algèbre linéaire, plus particulièrement relatifs à la résolution de grands systèmes linéaires. On se place ici dans un cadre applicatif industriel où les matrices manipulées sont creuses (discrétisation par éléments finis) et où les contraintes numériques nécessitent l'utilisation de solveurs directs réputés pour leur robustesse vis à vis de ces problèmes numériques. Cependant, la mise en œuvre d'une chaîne complète hautes performances pour la résolution par méthode directe de très grands systèmes linéaires creux sur des calculateurs parallèles modernes ayant un grand nombre de processeurs est une entreprise longue et complexe compte tenu du nombre et de la difficulté des problèmes à résoudre (optimisation de la numérotation des inconnues en utilisant des techniques provenant de la théorie des graphes, optimisation de la distribution des données et des calculs sur les processeurs pour un problème irrégulier, optimisation des échanges de données et de la communication, ...).

Ce travail de thèse s'inscrit dans cette problématique. Il a été réalisé dans le cadre d'un contrat de recherche et de développement passé entre le LaBRI UMR 5800 (Equipe ALiENor) et le CEA/CESTA (Contrat 7V1555AC). Ce contrat concernait initialement (ce travail est en voie d'extension à un code d'électromagnétisme) la parallélisation du code OSSAU du CEA/CESTA qui est un code industriel de mécanique des structures, en deux ou trois dimensions d'espace. C'est un code de calcul par éléments finis qui résout les équations de la mécanique des milieux continus en statique ou en dynamique (quand les effets d'inertie sont non négligeables) ; il calcule donc la réponse d'une structure à diverses sollicitations (forces ponctuelles, pressions, dilatations thermiques, accélérations diverses). Ces sollicitations peuvent conduire à l'apparition de plasticité et/ou de grands déplacements rendant le problème non linéaire ; on est alors amené à résoudre de nombreuses fois de grands systèmes linéaires creux mal conditionnés.

Le premier objectif de ce contrat était donc de concevoir et de réaliser une bibliothèque algorithmique complète hautes performances industrialisable pour la résolution par méthode directe de tels grands systèmes linéaires creux sur des calculateurs parallèles à mémoire distribuée. Le deuxième objectif était de réaliser l'intégration de ces outils dans une chaîne pré-industrialisée, et de la valider sur le code OSSAU pour des problèmes dépassant le million d'inconnues en 3D.

Cette thèse est décomposée en trois parties. Dans la première partie qui constitue en fait une introduction détaillée, nous présentons le problème mathématique traité en faisant un rappel de la méthode des éléments finis appliquée à l'élasticité (chapitre 1). Toujours dans cette première partie, nous rappelons le cadre général pour la mise en œuvre d'un solveur direct parallèle hautes performances (chapitre 2).

Dans la deuxième partie de la thèse, nous présentons l'ensemble des choix retenus pour la chaîne logicielle proposée par l'ensemble de l'équipe ALiENor du LaBRI (logiciel EMILIO), et nous donnons dans les annexes un grand nombre de mesures de performances sur des tests de référence montrant la qualité des outils réalisés (chapitre 3). Les deux chapitres suivants (chapitres 4 et 5) sont consacrés à deux étapes spécifiques de la chaîne logicielle et qui sont la distribution des éléments du maillage et l'assemblage parallèle de la matrice, ces deux chapitres constituant la première contribution majeure de ce travail de thèse qui est de nature algorithmique.

Enfin, dans la dernière partie, nous présenterons le résultat de la deuxième contribution de ce travail qui a consisté à intégrer tous ces outils dans un enchaînement opérationnel portable, bien articulé avec les parties du code existant traitant la physique du problème, et que nous avons évalué sur un calculateur parallèle de type IBM SP2. Nous détaillons ainsi 3 études de cas provenant de problèmes industriels réels issus du CEA/CESTA (chapitres 7, 8 et 9).

Une conclusion donnera finalement le bilan de ce travail ainsi que les perspectives à moyen et à long termes.

Première partie

Cadre de l'étude

Chapitre 1

Le problème mathématique

1.1 Rappel de la méthode des Eléments Finis appliquée à l'élasticité

1.1.1 Elasticité linéaire

Le théorème des travaux virtuels exprime l'égalité sur une structure donnée Ω et sous un déplacement virtuel δu, entre le travail des forces internes et celui des chargements externes. On a

$$\int_\Omega \underbrace{(\underbrace{\delta\epsilon^t\sigma}_{contraintes} - \underbrace{\delta u^t b}_{forces\ réparties})}_{travail\ des\ forces\ internes} dv = \underbrace{\int_\mu \underbrace{\delta u^t P ds}_{forces\ de\ pression} + \underbrace{\delta u^t q}_{forces\ ponctuelles}}_{travail\ des\ chargements\ externes} \quad (1)$$

où
δu^t est le vecteur transposé du vecteur déplacements virtuels,
$\delta\epsilon^t$ est le vecteur transposé du "vecteur" déformations,
σ est le "vecteur" contraintes,
q représente les forces ponctuelles appliquées.

On supposera les forces réparties b (accélérations, ...) et les pressions P nulles pour la suite. En 3D, élément fini volumique, le vecteur déplacement u est formé de 3 composantes

$$\left\{ \begin{array}{c} u_1 \\ u_2 \\ u_3 \end{array} \right\}$$

qui sont les translations selon les 3 axes.
On réalise une discrétisation de la structure en éléments finis et donc, en tout point, le vecteur déplacements va s'exprimer à l'aide des fonctions de forme et des déplacements des nœuds de la discrétisation. Ainsi pour la composante u_1

$$u_1(x) = \sum_{i=1}^{N} N_i(x) \sqcup_1^i$$

où $\sqcup_1^i$ est le déplacement du nœud i dans la direction 1, et $N_i(x)$ la valeur de la fonction de forme associée au nœud i et calculée à la position x.

Sous forme matricielle cela donne

$$\overrightarrow{u}(x) = \left\{ \begin{array}{c} u_1 \\ u_2 \\ u_3 \end{array} \right\} = \underbrace{\begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & \cdots \\ 0 & N_1 & 0 & 0 & N_2 & 0 & \cdots \\ 0 & 0 & N_1 & 0 & 0 & N_2 & \cdots \end{bmatrix}}_{N(x)} \underbrace{\begin{bmatrix} \sqcup_1^1 \\ \sqcup_2^1 \\ \sqcup_3^1 \\ \sqcup_1^2 \\ \vdots \end{bmatrix}}_{\overrightarrow{\sqcup}} \tag{2}$$

Le vecteur $\overrightarrow{\sqcup}$ est de taille $3n$ si n est le nombre de nœuds de la structure. $\overrightarrow{\sqcup}$ sera appelé vecteur déplacements nodaux. Sa connaissance permettra de déduire par la formule ci-dessus les déplacements en tout point de la structure. On peut réécrire la formule 2 en

$$\overrightarrow{u}(x) = N(x) \overrightarrow{\sqcup}. \tag{3}$$

En théorie des petites déformations, on déduit le "vecteur" déformations du vecteur déplacements par

$$\overrightarrow{\varepsilon} = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{33} \end{bmatrix} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad i = 1,3 \quad j = 1,3\ . \tag{4}$$

Soit en appliquant la formule 3

$$\overrightarrow{\varepsilon}(x) = \ell(\overrightarrow{u}(x)) = \ell(N(x)) \overrightarrow{\sqcup} = B \overrightarrow{u}.$$

La matrice B est une matrice $6 \times (3n)$ obtenue par application de l'opérateur ℓ (de type dérivation) à la matrice de fonctions de forme. En portant cela dans l'équation 1 on obtient

$$\delta u^T \int_\Omega B^T \sigma dv = \underbrace{\delta u^T q}_{F} \ . \tag{5}$$

En élasticité linéaire les contraintes sont liées aux déformations par

$$\sigma = D\varepsilon \ . \tag{6}$$

On supposera qu'il n'y a pas de déformation thermique, ni de contrainte initiale. On arrive donc au système linéaire suivant

$$K\overrightarrow{\sqcup} = F \text{ avec } K = \int_\Omega B^T D B dv = \int_\Omega \ell^T(N) D \ell(N) dv \ .$$

La matrice K est appelée matrice de rigidité. Du fait de la présence de 3 inconnues par nœud, la matrice K est formée de blocs 3×3. Le bloc $i \times j$ correspond à l'interaction de l'opérateur ℓ^t appliqué à la fonction de forme N_i avec l'opérateur ℓ appliqué à N_j. On a donc la propriété habituelle des matrices éléments finis à savoir le caractère creux. Le bloc $K_{i\times j}$ est non nul uniquement si les supports des fonctions de forme N_i et N_j ont une intersection non nulle : le support de N_i (resp. N_j) étant limité aux éléments finis dont i (resp. j) est un sommet, les blocs $K_{i,j}$ sont non nuls si les nœuds i et j appartiennent à un même élément.
Tant que la structure reste élastique, la matrice D est uniquement fonction du matériau présent dans la structure à l'endroit où on écrit l'équation 6. Les éléments finis de la structure ont donc un attribut matériau qui désigne quelles sont les valeurs du module de Young E et du coefficient de Poisson ν qui participent à la matrice D. Dans le cas 3D volumique qui nous intéresse ici, la matrice D (6×6) s'écrit pour un matériau isotrope

$$D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \nu(1-\nu) & \nu(1-\nu) & 0 & 0 & 0 \\ & 1 & \nu(1-\nu) & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ SYM & & & \frac{1-2\nu}{2(1-\nu)} & 0 & 0 \\ & & & & \frac{1-2\nu}{2(1-\nu)} & 0 \\ & & & & & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix}$$

1.1.1.1 Assemblage

L'intégrale sur le volume Ω peut s'exprimer à partir des intégrales sur les éléments finis e qui discrétisent Ω.

$$K\overrightarrow{\sqcup} = \int_{\Omega} \ell^T(N) D \ell(N) dv \overrightarrow{\sqcup} = \sum_e \int_e \ell^T(N) D \ell(N) dv \overrightarrow{\sqcup} \ . \tag{7}$$

Vue la propriété des fonctions de forme N_i d'être non nulles uniquement sur les éléments dont i est un sommet, une grande partie de la matrice N est nulle quand on intègre sur l'élément fini e et donc 7 se réduit à

$$K\overrightarrow{\sqcup} = \sum_e \underbrace{\int_e B_e^T D B_e \, dv}_{K_e} \overrightarrow{\sqcup} e \ . \tag{8}$$

La notation B_e signifie "restriction de la grande matrice B $(6 \times (3 \times n))$ à sa partie utile dans l'élément fini e". B_e a pour taille $6 \times (3 \times n_e)$ avec n_e nombre de nœuds de l'élément fini. $\overrightarrow{\sqcup}_e$ est la restriction du vecteur déplacements nodaux complet $\overrightarrow{\sqcup}$ aux déplacements nodaux de l'élément e. On a donc

$$\overrightarrow{K} = \sum_e K_e \ . \tag{9}$$

La signification de $\sum$ dans l'équation 9 est alors un peu particulière. Il s'agit bien de sommer les matrices K_e dans la matrice K mais comme K_e est de taille bien inférieure à K, cette opération n'est faite qu'après avoir réalisé l'extension de K_e à la taille de K (i.e. avoir placé la "matrice restreinte K_e" au bon endroit dans la matrice K). Cette opération s'appelle "assemblage".

1.1.1.2 Calcul des matrices de rigidité élémentaires K_e

Pour calculer les fonctions de forme, on a l'habitude de se ramener sur un élément fini de référence. Par exemple pour un élément fini e qui est un hexaèdre quelconque de Ω, on se ramène sur le cube $[-1, 1] \times [-1, 1] \times [-1, 1]$.

On dit qu'un élément fini est "isoparamétrique" si on utilise pour ce "mapping" les mêmes fonctions de forme que celles utilisées pour interpôler les fonctions inconnues. Dans ce cas, à l'intérieur de chaque élément fini e

$$\overrightarrow{\sqcup}_e = \left\{ \begin{array}{l} u_{1_e}(x, y, z) = \sum_1^{n_e} N_i(x, y, z) u_1^i \\ u_{2_e}(x, y, z) = \sum_1^{n_e} N_i(x, y, z) u_2^i \\ u_{3_e}(x, y, z) = \sum_1^{n_e} N_i(x, y, z) u_3^i \end{array} \right\}$$

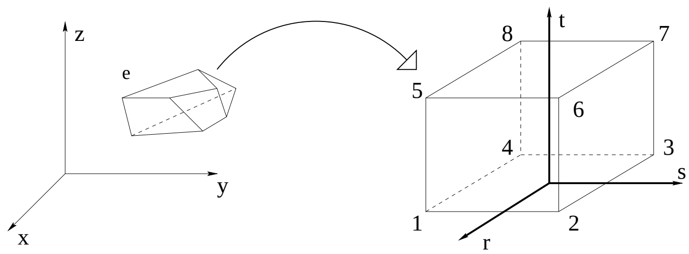

FIG. 1.1: Un hexaèdre à interpolation linéaire et Elément de référence

où par exemple, u_1^i est le déplacement en x du nœud i de l'élément fini e. Dans e, les coordonnées x,y,z se déduisent alors de celles des sommets de e par

$$\begin{aligned} x &= \textstyle\sum_1^{n_e} N_i(r,s,t)x^i \\ y &= \textstyle\sum_1^{n_e} N_i(r,s,t)y^i \\ z &= \textstyle\sum_1^{n_e} N_i(r,s,t)z^i \end{aligned}$$

x^i, y^i, z^i étant les coordonnées du nœud i.

Il est plus facile d'écrire les fonctions de forme sur l'élément de référence (i.e. en termes de r, s, t). Par exemple, pour le cube à 8 nœuds interpolation linéaire (voir figure 1.1), on a pour le nœud 6 situé en $r = 1$, $s = 1$, $t = 1$

$$N_6(r,s,t) = \tfrac{1}{8}(1+r)(1+s)(1+t)\ .$$

Les autres se déduisent aisément. Pour le même cube (à 20 nœuds) en interpolation quadratique de la famille de Serendipity, avec pour nœuds les 8 sommets du cube plus les nœuds milieux des arêtes, on a

$$N_6(r,s,t) = \tfrac{1}{8}(1+r)(1+s)(1+t)(r+s+t-2)\ .$$

Pour un nœud milieu des arêtes, par exemple le noeud 9 qui serait situé au milieu de l'arête (6,7) avec $r = 0$, $s = 1$, $t = 1$, on aurait alors

$$N_9(r,s,t) = \tfrac{1}{4}(1-r^2)(1+s)(1+t)\ .$$

Ces fonctions de forme respectent bien les règles habituelles, à savoir

$$\begin{gathered} N_i = 1 \text{ pour le sommet } i\,;\ N_i = 0 \text{ pour les autres sommets} \\ \text{et } \textstyle\sum_i N_i(r,s,t) = 1 \quad \forall r,s,t \in [-1,+1]\ . \end{gathered}$$

A partir de ceci, il reste à calculer

$$K_e = \int_e B^T DBdv \ .$$

soit plus précisément (comme il n'y a pas d'ambiguité, on a abandonné la notation B_e au profit de B)

$$\int_x \int_y \int_z B^T(N(x,y,z)) \ D \ B(N(x,y,z)) \ dx \ dy \ dz \ .$$

On utilise comme changement de variable le "mapping" défini ci-dessus qui permet de se ramener sur l'élément de référence

$$dx \ dy \ dz = detJ \ dr \ ds \ dt$$

J étant la matrice Jacobienne du "mapping" défini ci-dessus. On a alors

$$K_e = \int_{r=-1}^{r=1} \int_{s=-1}^{s=1} \int_{t=-1}^{t=1} B^T(N(r,s,t)) \ D \ B(N(r,s,t)) \ detJ \ dr \ ds \ dt \ .$$

Ces intégrations sur l'élément fini de référence sont alors beaucoup plus faciles à réaliser. Arguant du caractère polynomial de N, maintenu dans B qui est un opérateur de dérivation, on a un intégrande de type polynomial (idem pour $det \ J$ quand l'élément fini est isoparamétrique). Les intégrales sont alors calculées par quadrature de Gauss produit dans chaque direction de l'espace.

Il vient alors

$$K_e = \sum_{i=1}^{p} \sum_{j=1}^{p} \sum_{k=1}^{p} w_i w_j w_k \ B_T(N(r_i, s_j, t_k)) \ D \ B \ detJ(r_i, s_j, t_k)$$

les r_i, s_j, t_k étant les abscisses des points de Gauss, et les w_i, w_j, w_k les poids associés. Pour un cube à 8 nœuds, on peut prendre $p = 2$, et pour un cube à 20 nœuds, il faut prendre $p = 3$.

Ceci nous donne une idée du coût de calcul de la matrice élémentaire K_e. L'intégrande dans le cas du cube à 20 nœuds est à évaluer 27 fois : $3 \times 3 \times 3$. Il comprend le calcul de la matrice Jacobienne et de son déterminant, le calcul de la matrice B ($6\times$ (3×20)), sa multiplication par la matrice D (6×6) et la multiplication par B^T ((3×20)$\times6$).

1.1.2 Problèmes non linéaires

Concernant les méthodes décrites dans cette partie, on pourra consulter les ouvrages de référence [69, 70].

1.1.2.1 Méthodes de résolution

Ce sont les cas où la matrice K et où le second membre F varient en fonction du déplacement. On a donc à résoudre des problèmes du type

$$\psi(\overrightarrow{\sqcup}) = 0 \text{ avec } \psi(\overrightarrow{\sqcup}) = P(\overrightarrow{\sqcup}) - F(\overrightarrow{\sqcup}) = K(\overrightarrow{\sqcup})\overrightarrow{\sqcup} - F(\overrightarrow{\sqcup})\ . \tag{10}$$

On supposera dans la suite que F ne dépend pas de $\overrightarrow{\sqcup}$. Pour résoudre ceci, on peut utiliser plusieurs méthodes de type itératif.

1.1.2.2 Méthode du point fixe ("Rigidité sécante")

On part d'une estimation initiale du vecteur de déplacements nodaux $\overrightarrow{\sqcup}_0$ (qui peut être identiquement nul) et on réalise l'algorithme suivant :
A partir de $\overrightarrow{\sqcup}_0$, on calcule $\overrightarrow{\sqcup}_1$ tel que

$$K(\overrightarrow{\sqcup}_0)\overrightarrow{\sqcup}_1 = \overrightarrow{F}$$

puis $\overrightarrow{\sqcup}_2$ tel que

$$K(\overrightarrow{\sqcup}_1)\overrightarrow{\sqcup}_2 = \overrightarrow{F}$$

et ainsi de suite. On peut faire apparaître un résidu en écrivant

$$\overrightarrow{\sqcup}_{i+1} = \overrightarrow{\sqcup}_i + \Delta\overrightarrow{\sqcup}_i$$

puis

$$K(\overrightarrow{\sqcup}_i)\Delta\overrightarrow{\sqcup}_{i+1} = F - K(\sqcup_i)\sqcup_i = \underbrace{F - P(\sqcup_i)}_{R_i} = -\psi(\sqcup_i)$$

R_i étant appelé résidu à l'itération i.

1.1.2.3 Méthode de Newton ("Rigidité tangente")

On fait le développement limité suivant

$$\psi(\sqcup_{n+1}) = \psi(\sqcup_n) + \tfrac{d\psi}{d\sqcup}(\sqcup_n)\Delta\sqcup_{n+1} + \ldots$$

Pour compacter l'écriture, nous noterons désormais $(\frac{d\psi}{d\sqcup})_n$ à la place de $\frac{d\psi}{d\sqcup}(\sqcup_n)$.
Si on veut que $\sqcup_{n+1}$ soit solution de

$$\psi(\sqcup_{n+1}) = 0$$

alors

$$(\tfrac{d\psi}{d\sqcup})_n \Delta\sqcup_{n+1} = -\psi(\sqcup_n)$$

avec

$$\tfrac{d\psi}{d\sqcup} = \tfrac{dP}{d\sqcup} = \tfrac{dK}{d\sqcup} \sqcup + K \ .$$

En général, on ne sait pas dériver directement la matrice de rigidité K, mais on arrive à exhiber directement $\frac{dP}{d\sqcup}$ que l'on appelle $K_T(\sqcup)$ matrice de rigidité tangente. Le schéma s'écrit alors

$$K_T(\sqcup_n)\Delta\sqcup_{n+1} = -\psi(\sqcup_n) = \underbrace{F - P(\sqcup_n)}_{R_n} \ . \tag{11}$$

1.1.2.4 Méthodes incrémentales

Les 2 méthodes ci-dessus n'ayant pas un rayon de convergence infini, l'idée est venue de "saucissonner" le chargement F et de l'appliquer progressivement. Soit alors à résoudre

$$P(\sqcup) - \lambda F = 0, \quad \lambda \text{ variant de 0 à 1}$$

ce qui conduit à

$$\tfrac{dP}{d\sqcup}\tfrac{d\sqcup}{d\lambda} - F = 0$$

soit

$$K_T \tfrac{d\sqcup}{d\lambda} = F$$

Pour résoudre cette équation différentielle, on a 2 alternatives :

- une méthode explicite, ce qui conduit à un petit incrément en λ. Par exemple, on peut considérer la méthode d'Euler pure

  $$K_T(\sqcup_n)\Delta\sqcup_{n+1} = F\Delta\lambda_{n+1} \ . \tag{12}$$

  Cette méthode est peu utilisée à cause des toutes petites valeurs $\Delta\lambda$ à employer (dûes à la condition C.F.L.), qui amène à un trop grand nombre de résolutions.

– une méthode implicite

$$K_T(\sqcup_{n+1})\Delta\sqcup_{n+1} = F\Delta\lambda_{n+1} \ .$$

Ce système est strictement équivalent à 10 mais en ayant "saucissonné" le chargement. Au lieu de chercher à passer directement de

$$(\overrightarrow{\sqcup} = 0, \overrightarrow{F} = 0) \text{ à } (\overrightarrow{\sqcup}(F_{total}), \overrightarrow{F}_{total})$$

on utilise un cheminement

$$(\overrightarrow{\sqcup} = 0, \overrightarrow{F} = 0), (\overrightarrow{\sqcup}_1, \lambda_1 \overrightarrow{F}), (\overrightarrow{\sqcup}_2, \lambda_2 \overrightarrow{F}), \cdots, (\overrightarrow{\sqcup}_{total}, \overrightarrow{F}_{total}) \ .$$

On utilisera de grands incréments $\Delta\lambda$ quand le problème est faiblement non linéaire, de plus petits déplacements quand le problème devient fortement non linéaire. L'algorithme incrémental couplé de Newton s'écrit donc pour l'incrément $n+1$ (après convergence de l'incrément n)

$$K_T(\sqcup_{n+1}^{(0)})\Delta\sqcup_{n+1}^{(1)} = F\Delta\lambda_{n+1}$$

où l'indice supérieure (i) correspond à l'indice d'itération dans l'incrément courant ($\sqcup_{n+1}^{(0)} = \sqcup_n$ convergé à l'incrément précédent)

$$\sqcup_{n+1}^{(1)} = \sqcup_{n+1}^{(0)} + \Delta\, \sqcup_{n+1}^{(1)} \ .$$

On calcule alors le résidu

$$R_{n+1}^{(1)} = \lambda_{n+1}F - P(\sqcup_{n+1}^{(1)})$$

puis on résout

$$K_T(\sqcup_{n+1}^{(1)})\Delta\sqcup_{n+1}^{(2)} = R_{n+1}^{(1)} \tag{13}$$

et ainsi de suite.
Donc à l'itération i, on a à résoudre

$$K_T(\sqcup_{n+1}^{(i)})\Delta\sqcup_{n+1}^{i+1} = R_{n+1}^{(i)} \ . \tag{14}$$

La convergence est obtenue soit quand $\Delta\sqcup_{n+1}^{i+1} = 0$, soit quand $R_{n+1}^{(i)} = 0$.
Une variante consiste à figer la matrice tangente pendant quelques itérations n_f, ce qui conduit à

$$K_T(\sqcup_{n+1}^{(i)})\Delta\sqcup_{n+1}^{i+j} = R_{n+1}^{i+j-1} \quad j = 1, n_f \ . \tag{15}$$

L'utilisation de méthodes directes pour résoudre le système linéaire permet alors de *ne triangulariser la matrice K_T que de temps en temps, et de ne faire que des descentes-remontées pendant les n_f itérations* où l'on a décidé de figer la matrice tangente. Même si le nombre d'itérations total augmente, le coût global peut être nettement inférieur.

1.1.2.5 Cas de la plasticité

Dans ce cas, c'est la relation 6 qui perd son caractère linéaire. En 1D, cette relation devient

$$\sigma = E\varepsilon \ .$$

σ et ε sont maintenant des scalaires. Pour des ε petits, la relation entre ε et σ reste linéaire, puis apparaît le phénomène de plastification qui se traduit par la figure 1.2.

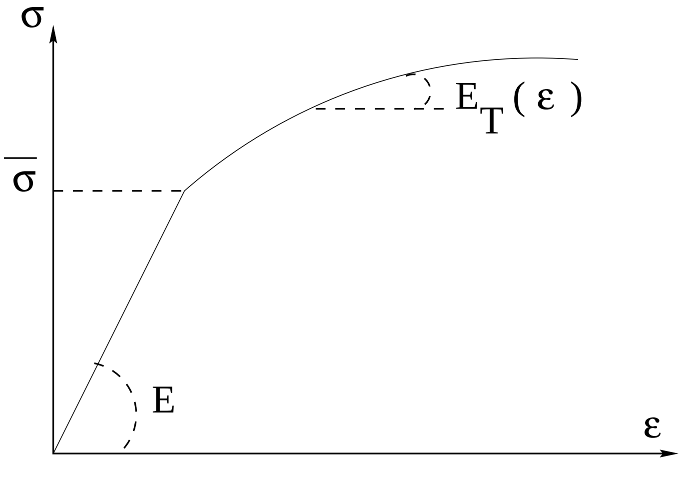

FIG. 1.2: Le cas de la plasticité

Jusquà $\overline{\sigma}$, on a $\sigma = E\varepsilon$ ou $\Delta\sigma = E\Delta\varepsilon$, et au delà on a $\Delta\sigma = E_T(\varepsilon)\Delta\varepsilon$. Dans ce cas, on ne sait pas exhiber facilement la matrice de rigidité sécante ; par contre on peut exhiber

$$K_T = \frac{dP}{d\sqcup} = \int_\Omega B^T \frac{d\sigma}{d\varepsilon}\frac{d\varepsilon}{d\sqcup} dv = \int_\Omega B^T E_T(\varepsilon) B dv \ .$$

Bien sûr, ici encore la matrice K_T est obtenue par sommation des matrices élémentaires K_{T_e}, le calcul des $E_T(\varepsilon)$ étant alors ramené au niveau des points de Gauss dans l'élément fini en question

$$K_e = \sum_i \sum_j \sum_k \omega_i \omega_j \omega_k \ B_T \ E_T(\varepsilon(r_i, s_j, t_k)) \ B \ det \ J \ .$$

Le résidu apparaissant au second membre

$$R_n = F - P(\sqcup_n)$$

se décline lui aussi pour la partie $P(\sqcup_n)$ au niveau élément fini

$$P(\sqcup_n) = \sum_e P_e(\sqcup_n) \ avec \ P_e(\sqcup_n) = \int_e B^T \sigma(\sqcup_n) dv \ .$$

Tant que l'on reste dans le domaine linéaire, il n'y a pas de problème. Dès que l'on franchit $\overline{\sigma}$, limite de plasticité, les choses deviennent plus complexes. Il faut suivre de manière incrémentale la courbe $\sigma(\varepsilon)$. Cela se réalise par des techniques de projection que l'on n'expliquera pas ici mais que l'on peut résumer schématiquement en les étapes suivantes :

- on connaît au point (x,y,z) ε_{n-1} et donc σ_{n-1},
- la résolution de 11 nous donne $\Delta \overrightarrow{\sqcup}$,
- on en déduit $\Delta\varepsilon$ et $\Delta\sigma$ aux points d'intégration de chaque élément fini,
- on projette $\sigma + \Delta\sigma$ sur le convexe de plasticité (si besoin : cas des contraintes non admissibles),
- on obtient $\sigma(\varepsilon)$,
- et on calcule finalement le résidu.

Ici encore ces calculs se font au niveau des points d'intégration à l'intérieur de chaque élément fini. Il apparaît clairement que le calcul de la matrice de rigidité élémentaire ou du résidu élémentaire est plus complexe dans un élément fini après plastification qu'avant quand son comportement reste linéaire. Cela pourra avoir de l'influence sur la parallélisation de la phase d'assemblage. Le problème en multi-D est beaucoup plus compliqué et le lecteur pourra se reporter à [70] et [47] pour avoir une idée des différentes techniques pouvant être alors utilisées.

1.1.2.6 Cas des grands déplacements : non linéarité géométrique

La formule 4 liant les déformations aux déplacements n'est valable que lorsque ceux-ci restent petits. Au delà (et sans rentrer dans des détails trop techniques), on a des formules du type

$$\varepsilon = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\partial u_\alpha}{\partial x_i} \times \frac{\partial u\alpha}{\partial x_j}\right) .$$

On a alors

$$\begin{aligned} d\varepsilon = \overline{B}d\sqcup = [B_0 + B_1(\sqcup)]d\sqcup \Rightarrow \psi = \int \overline{B}^T \sigma dV \\ \Rightarrow d\psi = \int_V d\overline{B}^T \sigma dv + \int_V \overline{B}^T d\sigma dv = K_T d\sqcup \\ \Rightarrow d\psi = \underbrace{\int_V dB_1^T \sigma dv}_{K_\sigma d\sqcup} + \int_V [B_0 + B_1]^T D[B_0 + B_1] dv d\sqcup \ . \end{aligned}$$

On trouve en définitive

$$K_T = K_L + K_\sigma + \underbrace{B_0^T DB_1 + B_1^T DB_0 + B_1^T DB_1}_{K_1}$$

où

K_L est la matrice de rigidité linéaire classique,

K_σ dépend de l'état des contraintes de la structure,

K_1 dépend des déplacements de celle-ci.

La non-linéarité en grands déplacements peut être combinée avec la plasticité. C'est le cas des études du challenge de Cologne présentées au chapitre 7. Contrairement à la plasticité pure, les calculs supplémentaires à faire par rapport au cas linéaire sont ici équi-répartis sur l'ensemble des éléments finis de la structure.

1.1.3 Choix de la méthode de résolution du système linéaire

Quand le problème est globalement linéaire, on a à résoudre une seule fois le système

$$K\sqcup = F \ . \tag{16}$$

Quand le problème est non linéaire, on a à résoudre pour chaque incrément des systèmes du type 14 jusqu'à convergence. Les systèmes de type 14 sont bien de même nature que 16. Pour ce qui est de la forme 15, il s'agit de multi-systèmes linéaires de même matrice, mais multi-seconds membres (ces seconds membres *dépendant successivement les uns des autres*). Mais ici encore le point de passage obligé est la résolution de systèmes de type 16.

1.1.3.1 Propriétés de ces systèmes linéaires

La matrice K est creuse : nous l'avons déjà expliqué. Elle est symétrique de par même son écriture (la matrice D est symétrique y compris en plasticité).

En linéaire, les constantes d'élasticité E et ν sont telles que D est définie positive. Si les éléments finis ont une forme correcte (pas de pointe de flèche, d'allongement énorme, de nœuds "milieux" mal placés, ...), la matrice K est elle-même définie positive.

En plasticité, on voit qu'à l'extrémité de la courbe de la figure 1.2, E_T (dans ce cas mono-dimensionnel) peut s'annuler. Cette annulation peut même arriver dès le passage de la limite $\overline{\sigma}$. Si tous les points d'intégration d'un élément fini se trouvent dans cette situation, la matrice élémentaire de cet élément fini n'est plus définie positive. Si tout un ensemble d'éléments d'une structure se trouve dans cette situation, il peut se créer ce que les mécaniciens appellent une "rotule plastique" : une partie de la structure ne présente plus aucune rigidité et un incrément infime de charge peut créer une réponse infinie en déplacement. La matrice $K_T(\sqcup)$ n'est plus alors définie positive. En général, on arrête à ce niveau les calculs de la structure.

Dans le cas de non linéarités dues aux grandes déformations, la matrice K_T de la formule 14 peut au cours de l'évolution du chargement ne plus être définie positive. C'est le cas bien connu de la coupole sphérique sur le pôle de laquelle, on impose une charge incrémentale. Si on trace la courbe déplacement du pôle par rapport au chargement, on obtient une courbe d'abord linéaire, puis le déplacement augmente plus rapidement puis pour un certain niveau de chargement, il y a "clocage", le déplacement saute brutalement alors que l'on augmente plus la charge (voir figure 1.3). Au point (U_*, F_*), clairement le système 14 n'est pas défini positif.

Certaines méthodes cherchent à suivre le comportement du déplacement dans la zone en pointillés (zone instable suivie d'un retour à la stabilité). Ces méthodes dites de pilotage en déplacements ont à résoudre plusieurs fois le système 14 dans des zones où il est presque singulier et, où en tout cas il n'est pas défini positif.
Même si faute de temps, la présente étude n'a pu être étendue à ce dernier type de problème, il faut s'attendre à une matrice K ayant des valeurs propres proches de 0 quand on veut traiter des problèmes à fort degré de plastification et (ou) à grande déformation.

De plus, ces matrices même dans le cas linéaire élastique, ne sont pas des M matrices (ni à diagonale dominante) quand la structure contient des éléments de la famille de Serendipity (tel l'hexaèdre à 20 nœuds décrit ci-dessus, élément pourtant

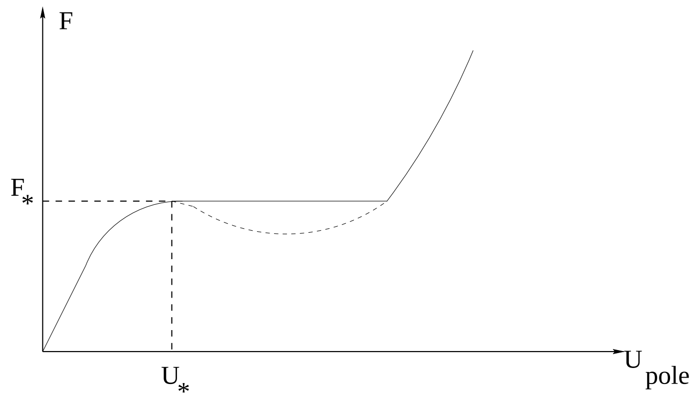

FIG. 1.3: La courbe déplacement

indispensable au rendu correct de certaines déformations [70]). Un préconditionneur de type décomposition de Cholesky incomplète (avec aucun remplissage) aura donc un comportement instable si on ne lui adjoint pas des artefacts supplémentaires [64].

1.1.3.2 Etude succincte du conditionnement

Au delà des appréciations qualitatives données dans le paragraphe précédent, nous avons voulu obtenir des résultats quantitatifs sur le conditionnement de ces systèmes linéaires.

Pour le cas test du Challenge de Cologne présenté au chapitre 7, nous avons calculé un conditionnement de 6 10^9 pour la matrice initiale (élastique). Plus tard au cours du chargement, l'apparition de la plasticité a fait chuter la plus petite valeur propre d'un facteur 5 conduisant à un conditionnement de 3 10^{10}.

Comme le conditionnement de la matrice initiale nous semblait un peu fort par rapport à l'évaluation réalisée par Manteuffel ($c = 10^8$ dans un cas apparemment similaire au nôtre avec les mêmes éléments finis de type hexaèdre à 20 nœuds famille de Serendipity, nombre de Degrés de Liberté comparable), nous avons voulu poursuivre l'investigation. La piste possible était le fait que tous les éléments finis de notre maillage présentaient un allongement proche de 20 (limite reconnue comme à ne pas dépasser en mécanique des structures). Nous avons donc réalisé un petit maillage de 3 éléments finis où les hexaèdres étaient des parallépipèdes rectangles d'extension maximale 2. Nous avons retenu des hexaèdres à 8 nœuds interpolation linéaire. Le conditionnement de la matrice est de $c = 687$. Si nous "applatissons" ce maillage pour arriver à une extension maximale de 20 pour l'un des éléments finis,

le conditionnement saute à $160\ 10^3$. Si maintenant nous passons à des éléments finis à 20 nœuds interpolation quadratique de Serendipity, il passe à $c = 1.63\ 10^7$ et cela pour 3 éléments finis seulement. Nous voyons donc que même pour des maillages restreints, nous sommes confrontés à des conditionnements très défavorables. Ce phénomène est accentué par l'apparition de la plasticité.
Enfin, les maillages que nous avons considérés sont mono-matériaux. La présence de matériaux de constantes élastiques ou élasto-plastiques très différentes ne peut que dégrader encore la situation. En plus de cette absence de "bonnes propriétés" et de ces mauvais conditionnements, les systèmes linéaires de forme 15 sont multi-seconds membres (apparaissant successivement). Ceci plaide aussi en faveur de l'utilisation de solveurs directs ; la triangularisation (opération la plus coûteuse) n'est réalisée que de temps en temps. C'est pour toutes ces raisons que traditionnellement les méthodes directes sont utilisées pour résoudre ce type de système linéaire. Dans notre cas, la matrice stockée après une renumérotation adéquate sous forme de profil colonne était résolue par une méthode de Cholesky avec pagination sur disque dans le code original.

Néammoins, au moment où l'on souhaite, grâce à l'utilisation des machines parallèles pouvoir traiter des problèmes de taille beaucoup plus importante (jusqu'au delà du million de degrés de liberté), il était bon d'analyser les progrès réalisés dans les méthodes itératives pour résoudre de tels systèmes.

1.1.3.3 Méthodes itératives

Manteuffel préconise [52] de stabiliser la décomposition de Cholesky incomplète sans remplissage $[ICG(0)]$, utilisée pour préconditionner un gradient conjugué, (cette décomposition incomplète pour être stable, demande la propriété de M-matrice) en "shiftant" le terme diagonal à l'aide d'un petit paramètre α. Néammoins, il ne traite des problèmes que de tailles limitées ($<$ 20.000 Degrés de Liberté), et on voit mal comment adapter l'évolution de α lorsque dans un problème de type plasticité, l'évolution de celle-ci dégrade de plus en plus le conditionnement du système à résoudre.

D'autres auteurs préconisent d'utiliser un préconditionneur de type Cholesky incomplet mais où les coefficients sont abandonnés non pas sur un critère d'existence ou non dans la matrice initiale, mais sur des critères de faiblesse de leurs valeurs absolues. Ces techniques ont l'inconvénient de créer un certain remplissage par rapport à la matrice initiale. Ces auteurs s'attaquent parfois avec ces techniques à des problèmes très mal conditionnés liés à la présence d'éléments dont l'allongement est très grand [38].

Pour pallier au défaut de la méthode précédente d'augmenter le remplissage, certains auteurs suggèrent d'ajouter des critères de place mémoire restante [67]. On trouve également une forte littérature sur l'utilisation de préconditionneurs hiérarchiques basés sur l'utilisation de maillages plus ou moins grossiers (cette littérature semble inspirée par les succès de la méthode multigrille dans d'autres domaines de la modélisation) [14, 18, 19, 54]. Certains abordent également le problème des multi-seconds membres cités ci-dessus [50]. De plus, on peut citer les travaux sur le préconditionnement de type "inverse approché" pour la méthode du Gradient Conjugué [13].

Néammoins, toutes les applications de ces approches itératives qui sont exposées, semblent limitées à un nombre intermédiaire de Degrés de Liberté (< 70000) et à l'élasticité linéaire. Nous craignions que la dégradation du conditionnement due à la plasticité et à un plus grand nombre de Degrés de Liberté, entraîne la nécessité de rajouter plus d'artefacts encore pour faire converger ces méthodes.

Une mention particulière doit être faite à la méthode FETI [25]. Cette méthode mixte un solveur direct et un solveur itératif en travaillant en 2 étapes. Une décomposition du domaine est réalisée. Les matrices de rigidité des sous-domaines sont condensées par une méthode directe sur les Degrés de Liberté des séparateurs. Le système linéaire résultant sur les séparateurs est ensuite résolu par un gradient conjugué préconditionné de manière très particulière. Il en résulte une méthode très fortement parallélisable, qui a permis de traiter des problèmes à des millions de Degrés de Liberté. Cette méthode a été appliquée à des problèmes non linéaires (non linéarité géométrique). A notre connaissance, elle n'a pas été appliquée à la plasticité.

Fort de cet argumentaire et de cet état des lieux, nous avons choisi de paralléliser un solveur direct, pour pouvoir résoudre des problèmes de type 16 de très grandes tailles, dans des temps CPU acceptables. L'introduction d'un tel solveur dans le code OSSAU du CEA/CESTA fait l'objet des chapitres ci-après.

Chapitre 2

Cadre général pour la mise en œuvre d'un solveur direct parallèle hautes performances

2.1 Introduction

Le domaine applicatif présenté au chapitre 1 a mis en évidence la nécessité d'utiliser des solveurs directs connus pour leur robustesse numérique. Ce type de résolution est en fait une brique de base pour traiter numériquement de nombreux problèmes de calcul scientifique. Les systèmes, qui apparaissent dans la discrétisation d'équations aux dérivées partielles par éléments ou volumes finis, sont de très grande taille pour les applications en vraie grandeur. Nous rappelons que nous visons ici des systèmes dépassant le million d'inconnues pour des problèmes 3D. Le parallélisme est une technique maintenant incontournable pour résoudre ces très grands systèmes. On pourra consulter [20] pour un état de l'art sur le domaine. Ce chapitre présente rapidement les idées centrales conduisant à l'implémentation de solveurs directs hautes performances pouvant traiter de tels systèmes. Le chapitre 3 reprendra chacun de ses points et précisera les choix retenus.

2.2 La renumérotation

Un des points essentiels propre aux méthodes directes sur les matrices creuses est le problème du *remplissage* [28]. En effet, la matrice factorisée de Cholesky L est plus pleine que la matrice initiale A de part la création de nouveaux termes non nuls au cours du processus de factorisation. Un outil fondamental pour étudier ce remplissage est le *modèle de graphe* [28] associé à l'élimination de Gauss. Ce remplissage étant directement lié à l'ordre d'élimination des inconnues, il s'agit donc de trouver une renumérotation de ces inconnues, c'est à dire une *renumérotation* [1, 28, 56, 57] *des sommets du graphe associé à la matrice initiale* qui minimise le remplissage et en même temps le nombre d'opérations à faire effectivement pour factoriser.

Le graphe non orienté G associé à une matrice A $n \times n$ symétrique ou à structure symétrique a n sommets, et il existe une arête (i, j) entre les sommets i et j si et seulement si $a_{ij} \neq 0$. Le graphe d'élimination est le graphe G^* associé à la matrice L triangulaire inférieure obtenue par factorisation ; G^* a bien sûr le même nombre de sommets que G mais a (beaucoup) plus d'arêtes, car il contient toutes les arêtes correspondant aux termes de remplissage. L'*arbre d'élimination* [51] T associé à la matrice factorisée est un arbre, au sens classique de la théorie des graphes, à n sommets, et il existe une arête entre i et j si et seulement si la ligne du premier terme non nul dans la colonne j dans la matrice factorisée L se trouve être i. L'arbre d'élimination joue un rôle fondamental pour la parallélisation des solveurs creux directs car il indique les dépendances dans les calculs : il ne peut y avoir de dépendances (et donc une séquentialité dans les calculs) entre deux éliminations d'inconnues que si les sommets associés sont sur une même branche de l'arbre d'élimination. Dans un cadre parallèle, le but est donc de trouver une renumérotation des sommets de G qui *minimise le remplissage* et qui *maximise l'indépendance dans les calculs de la factorisation* c'est à dire conduisant à des arbres d'élimination *larges et de faible hauteur*. Comme on le verra par la suite, les renumérotations performantes selon ces critères sont celles basées sur les techniques de "degré minimum" et de "dissections emboîtées" ; à noter que la numérotation classique de type Cuthill-McKee est à écarter car elle occasionne plus de remplissage et conduit en plus à des arbres d'élimination très longs avec donc une faible indépendance dans les calculs.

2.3 La factorisation symbolique

Une autre étape fondamentale est celle dite de *factorisation symbolique par bloc* [15] qui va permettre de calculer algorithmiquement la structure creuse par bloc de la matrice L à partir de celle de A. Le but est d'optimiser la *factorisation numérique* [5, 6, 7] en évitant la gestion effective durant les calculs des termes créés. En termes de graphe, cela revient à calculer, pour une numérotation donnée et pour une partition P des sommets associée correspondant à un découpage en *bloc-colonne* de la matrice, le *graphe quotient* G^*/P à partir de G et cette partition P. L'intérêt de l'algorithme de factorisation symbolique par bloc repose sur le fait que pour les partitions P considérées dans ce travail, les opérations de passage au quotient et d'élimination commutent, c'est à dire que l'on a l'égalité $(G/P)^* = G^*/P$. La complexité [15] en temps et en espace de cet algorithme croît alors comme le nombre total de blocs extra-diagonaux dans la structure ainsi construite pour L (voir figure 2.1) et dépend donc de la qualité de la numérotation vis à vis de la conservation du creux et de la partition P. Cette complexité est toujours très inférieure à celle de la factorisation numérique ce qui justifie son intérêt. L'arbre d'élimination est alors un *arbre d'élimination par bloc* et sa structure va décrire maintenant les dépendances dans les calculs par bloc du solveur.

Dans un cadre parallèle, cette étape permettra aussi la préparation des données pour la distribution optimisée sur les processeurs dans la phase de prétraitement.

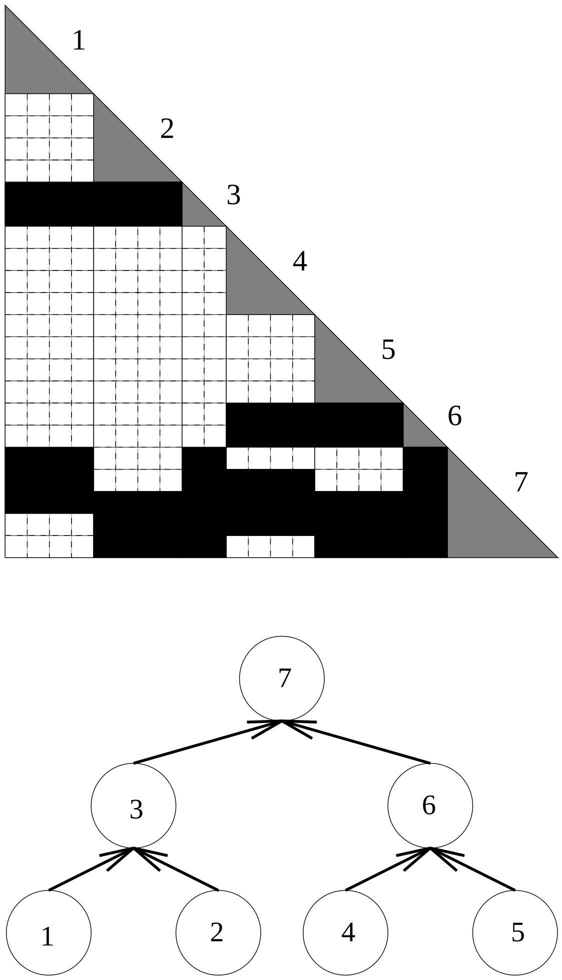

Fig. 2.1: Un exemple de matrice factorisée L structurée par bloc-colonne et d'arbre d'élimination par bloc. Chaque bloc-colonne est constitué d'un bloc diagonal et de plusieurs blocs extra-diagonaux denses.

2.4 Repartitionnement et distribution des blocs de la matrice

Suivant la nature de la matrice A, on distingue classiquement plusieurs types de factorisation :

- pour une matrice A symétrique définie positive, on peut utiliser la factorisation de Cholesky ($A = LL^t$) ou de Cholesky-Crout ($A = LDL^t$), avec ou sans pivotage numérique symétrique ;
- pour une matrice A non symétrique, on utilisera la factorisation LU avec pivotage non symétrique.

Le cadre parallèle retenu dans un premier temps pour ce travail a été celui des machines MIMD à mémoire distribuée, avec un modèle de programmation de type SPMD et utilisant une bibliothèque de communication de type MPI.

En fonction des propriétés de A, de la nature du solveur numérique retenu (mais aussi du degré d'hétérogénéité de la plateforme d'éxécution parallèle), on est amené à utiliser un ordonnancement statique (à précalculer aussi précisément que possible) et/ou dynamique des calculs par blocs suivant la capacité que l'on a à prédire à l'avance le comportement de ces calculs. Ce travail s'inscrit dans le cadre de la factorisation LDL^t de matrices symétriques définies positives *sans pivotage* implémentée sur des machines parallèles *homogènes* ; on se place donc dans une situation où il est intéressant de rechercher un *ordonnancement statique performant* [27, 43, 44, 58].

On doit aussi noter l'existence d'approches algorithmiquement différentes, pour prendre en compte la complexification liée à l'utilisation du pivotage numérique, telles que celle utilisée dans le logiciel SPOOLES développé à Boeing [12] et qui est basé sur une conception objets, et celle utilisée dans le logiciel MUMPS développé à Toulouse basé sur une technique multi-frontale pour une factorisation LU ou une factorisation LDL^T [3, 4].

Une autre phase de prétraitement des données va donc précéder l'exécution parallèle du solveur numérique. Ce prétraitement a pour rôle de calculer algorithmiquement un bon partitionnement et une bonne distribution des données ; ceci se fera à partir de la factorisation symbolique et par bloc. Le prétraitement va donc, en plus de la phase de renumérotation, calculer de manière statique une *régulation équilibrée* pour le solveur (critère d'équilibrage de charge des processeurs et prise en compte des contraintes de précédence) ; on utilisera pour cela la structure de l'arbre d'élimination par bloc. Le but sera d'exploiter les différents niveaux de parallélisme dans les calculs, à savoir :

1. le parallélisme à gros grain, induit par l'indépendance des calculs entre les

sous-arbres d'un même nœud de l'arbre d'élimination. On parlera aussi de parallélisme induit par le creux ;

2. le parallélisme à grain moyen, dû à la possibilité de raffiner la partition des inconnues en découpant un nœud de l'arbre d'élimination et en le distribuant sur plusieurs processeurs. C'est le parallélisme induit par une factorisation sur des blocs denses. On parlera aussi de parallélisme au niveau des nœuds.

3. le parallélisme à grain fin, ou micro-parallélisme. Celui-ci est obtenu en tirant partie du parallélisme interne d'un processeur lors du calcul sur les blocs (utilisation optimale du pipeline du processeur). Ce dernier niveau de parallélisme est fondamental pour obtenir de bonnes performances. Il repose, modulo *une bonne taille de blocs*, essentiellement sur l'utilisation des primitives BLAS3 [39, 40, 55, 60, 61, 63].

A noter que l'on essaiera aussi d'optimiser la localité des communications pour éviter des contentions dans le réseau du calculateur.

A partir de la structure de données par bloc calculée par la factorisation symbolique [15], la phase de prétraitement va donc calculer une bonne distribution des données sur les processeurs. On utilisera l'arbre d'élimination pour distribuer de manière équilibrée les blocs indépendants qui sont les plus bas dans l'arbre, puis on découpera les blocs situés dans les niveaux les plus hauts qui représentent de manière inhérente la plus grande masse de calcul. Les sous-blocs pourront alors être distribués par exemple de manière bloc cyclique pour utiliser le parallélisme contenu dans les calculs denses. On construit ainsi de manière statique une bonne régulation des calculs [24, 27, 59, 62].

Cette technique peut être utilisée selon un schéma de distribution 1D où l'on distribue de manière élémentaire des blocs-colonnes entiers. Schreiber [65] a montré que ce type de schéma limite la scalabilité des solveurs et qu'il faut alors avoir un schéma de distribution 2D où l'on distribue de manière élémentaire les blocs diagonaux et extra-diagonaux. C'est dans ce dernier cadre que l'on a atteint les meilleures performances connues à ce jour.

2.5 Résolution numérique

La phase de résolution numérique réalisera donc une factorisation parallèle de Crout LDL^T avec une distribution 1D des blocs de la matrice (c'est-à-dire par bloc-colonne) et/ou 2D (les blocs extra-diagonaux d'un bloc-colonne ne seront pas forcément tous distribués sur le même processeur).

Il existe essentiellement deux types d'approche qui sont l'approche *supernodale*, qui provient directement des algorithmes d'élimination par colonne réécrits par blocs et avec les variantes que l'on appelle *fan-in* ou *fan-out* [8, 9, 10, 11, 41, 55] et l'approche *multifrontale* [2, 16, 22, 23, 39, 40]. En ce qui concerne les méthodes supernodales qui constituent les méthodes retenues dans ce travail, l'approche fan-in ou "left-looking" consiste à répercuter sur le bloc-colonne courant les modifications des blocs-colonnes de gauche concernés, puis à calculer la valeur finale du bloc-colonne courant ; de manière symétrique, l'approche fan-out ou "right-looking" consiste à calculer définitivement le bloc-colonne courant et à répercuter ses contributions sur les blocs-colonnes de droite concernés. Dans un cadre parallèle, l'approche fan-in est plus performante car elle permet de limiter considérablement la masse de communications.

Ce survol rapide des différentes étapes pour la mise en œuvre d'un solveur parallèle direct étant fait, nous allons préciser chacune de ces étapes dans le chapitre suivant et les choix plus spécifiquement retenus. On décrira aussi le travail réalisé sur l'assemblage parallèle de la matrice de rigidité qui, en pratique pour notre application cible présentée au chapitre 1, est aussi une phase très coûteuse. L'ensemble constitue la chaîne logicielle EMILIO qui va être livrée au CEA/CESTA.

Deuxième partie

Description de la chaîne logicielle EMILIO

Chapitre 3

Introduction et étapes liées à la résolution

La chaîne logicielle EMILIO [33, 34, 35] développée au LaBRI par l'équipe ALiENor propose des solutions efficaces au problème de la résolution de grands systèmes linéaires creux par méthodes directes. L'ensemble des étapes algorithmiques disponibles dans la bibliothèque est présenté à la figure 3.1 dans le cadre de son utilisation pour des problèmes de mécanique des structures comme ceux exposés au chapitre 1. Les étapes parallèles sont situées à l'intérieur des cadres tandis que les flèches donnent tous les enchaînements possibles de ces étapes.

Cette chaîne logicielle est organisée en deux parties :

- une phase de prétraitement séquentiel réalisant les étapes suivantes : la renumérotation des nœuds du graphe d'éléments finis, la factorisation symbolique par blocs, le repartionnement et la distribution des blocs de la matrice (cette distribution pouvant être 1D et/ou 2D), la distribution des éléments du maillage (qui sert essentiellement pour l'assemblage parallèle),

- une phase parallèle réalisant l'assemblage parallèle (de la matrice K et du second membre) et la résolution parallèle du système.

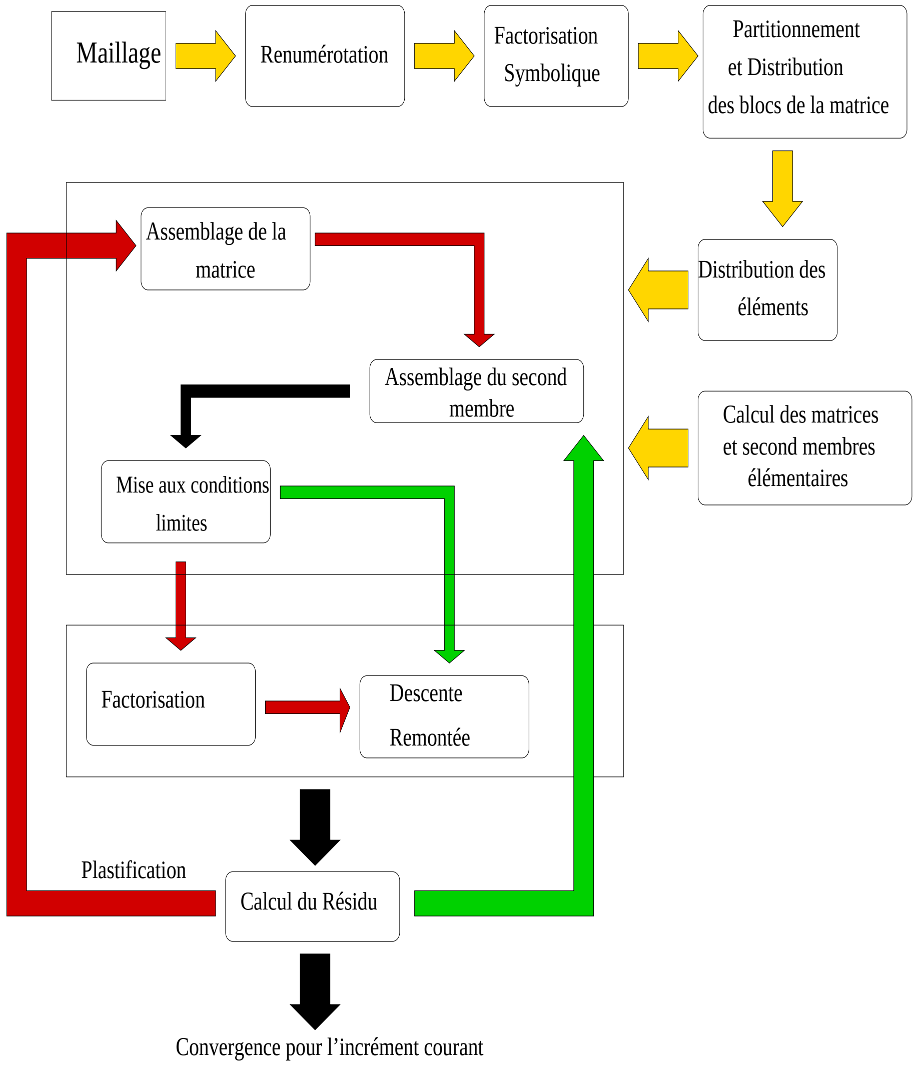

FIG. 3.1: L'enchaînement des différentes étapes

3.1 Enchaînement pour le prétraitement séquentiel des données

Le prétraitement séquentiel des données décrit en détails à la figure 3.2 regroupe les modules suivants :

- ORDER : ce module réalise la renumérotation des inconnues de la matrice afin tout d'abord de minimiser le remplissage et le nombre d'opérations lors de la factorisation, et conjointement de maximiser l'indépendance des calculs pour le traitement parallèle. Il utilise pour cela un couplage fin entre les techniques de *dissections emboîtées* et de *degré minimum*. ORDER manipule deux structures de données : en entrée, le graphe des nœuds, qui est construit à partir du maillage initial et converti au format reconnu par le logiciel SCOTCH. Ce graphe représente la structure de la matrice A. La deuxième structure de données est construite par le module ORDER et elle contient la nouvelle numérotation des sommets avec leur partition en blocs colonnes.

La méthodologie algorithmique qui est à la base de ce module ORDER (travaux de Francois Pellegrini en collaboration avec Patrick Amestoy et Jean Roman [56, 57]) est présentée dans le paragraphe 3.1.1.

– FAX : ce module réalise la factorisation symbolique par blocs de la matrice renumérotée et partitionnée en blocs colonnes. Il manipule trois structures de données : en entrée, il prend le graphe qui contient la structure de la matrice A à factoriser ainsi que la structure de données qui fournit la nouvelle numérotation des sommets et leur partition en blocs colonnes. Ces deux structures sont fournies par le module ORDER. Le module FAX construit alors la structure (symbolique) par blocs de la matrice factorisée. Des statistiques attachées à cette élimination, et relatives à l'encombrement mémoire et au nombre d'opérations à faire dans la factorisation/résolution sont également accessibles via une routine de ce module (voir fin du paragraphe 3.1.1).

– BLEND : ce module calcule algorithmiquement un "bon" repartitionnement, et donc une nouvelle partition des données, à partir de la factorisation symbolique par blocs effectuée précédemment. Ceci va consister à calculer de manière statique une régulation équilibrée pour le solveur (ordonnancement des calculs par blocs et des communications avec équilibrage de charge et prise en compte des contraintes de précédence). BLEND prend en entrée la structure de données issue de la factorisation symbolique par blocs et génère pour chaque processeur une structure de données qui contient la description des blocs distribués sur ce processeur ainsi que tous les renseignements dont ont besoin le solveur et le module d'assemblage : ordre de traitement des blocs colonnes, correspondance nœud - bloc colonne - processeur, etc ...

La méthodologie algorithmique qui est à la base de ce module BLEND (travaux de Pascal Hénon en collaboration avec Pierre Ramet et Jean Roman [43, 44]) est présentée dans le paragraphe 3.1.2.

– SACHA : ce module calcule une "bonne" distribution des éléments du maillage à partir de la distribution des blocs de la matrice calculée par le module BLEND. Cette distribution des éléments équilibre le calcul des matrices et/ou des seconds membres élémentaires ainsi que les mises à jour dans la matrice globale distribuée. SACHA prend en entrée la structure de données qui contient la distribution des blocs colonnes sur les processeurs ainsi que le maillage avec les nœuds renumérotés par le module Scotch. Il génére en sortie une structure de données qui contient des informations sur les éléments attribués à chaque processeur, ainsi que des informations sur les communications qui seront effectuées par ce processeur lors de la phase d'assemblage parallèle. Cette distribution des éléments est étudiée en détails au chapitre 4.

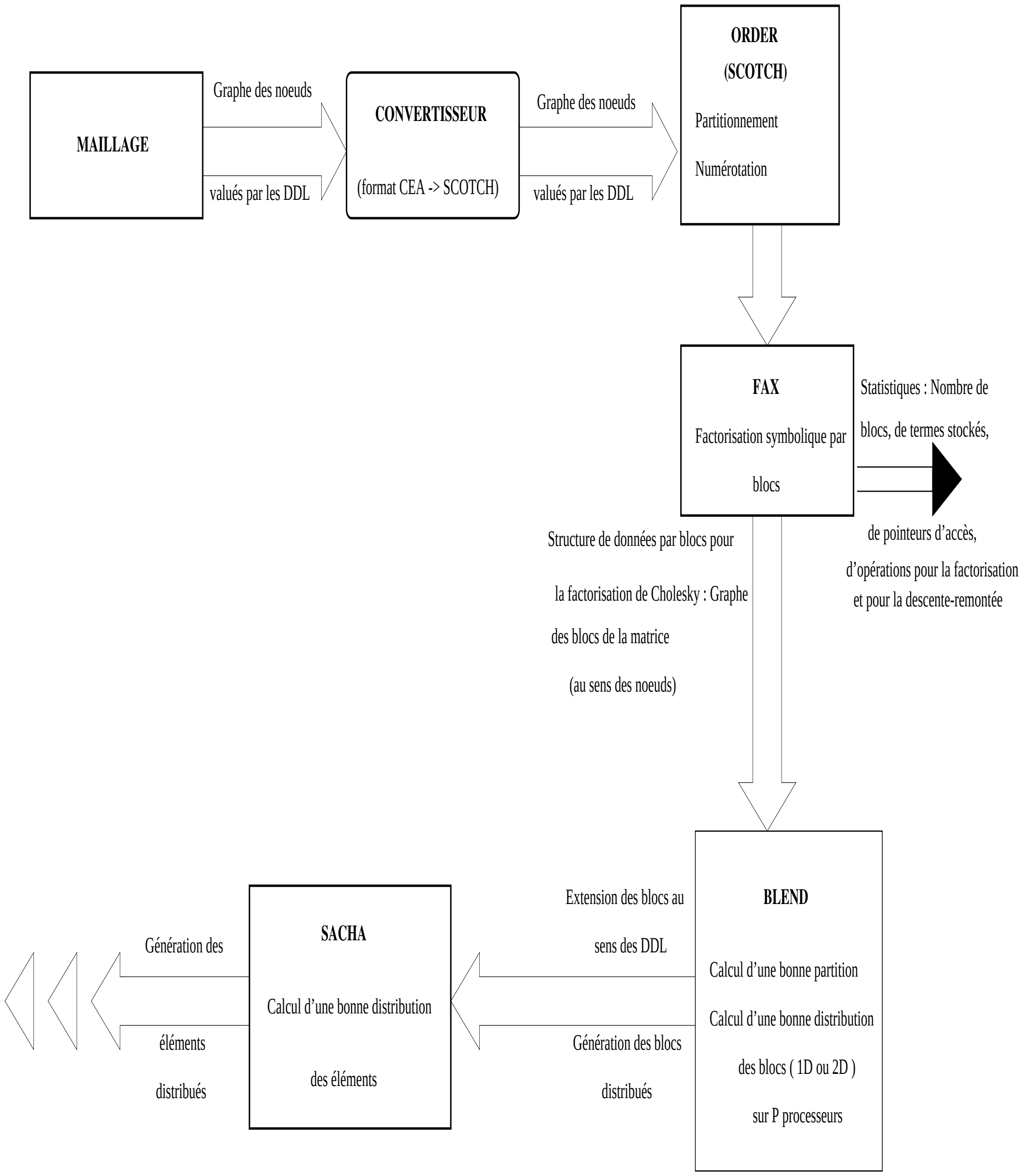

Fig. 3.2: Les étapes du prétraitement séquentiel

Un point important qu'il faut souligner ici est que les phases du prétraitement séquentiel qui consistent en la distribution des blocs de la matrice (module BLEND) et des éléments du maillage (module SACHA) obéissent à une *stratégie globale*. En effet, nous avons décidé de calculer tout d'abord une distribution en blocs optimisée pour la factorisation parallèle (les arguments sont clairement que la matrice factorisée est l'objet le plus encombrant en mémoire et que la factorisation est pour les grandes tailles l'étape la plus coûteuse en coût de calcul), puis de calculer une distribution des éléments induite par celle des blocs. Ce choix se justifie par les deux arguments suivants :

- tout d'abord par le fait qu'une distribution spécifique des blocs de la matrice pour l'assemblage parallèle nécessiterait dans l'absolu une *redistribution* pour la factorisation, ce qui est toujours une étape coûteuse en communications ;
- ensuite par le fait que la numérotation des nœuds du graphe d'éléments finis, qui utilise des techniques de partitionnement de graphes, induit une distribution des blocs de la matrice naturellement de bonne qualité pour réaliser un assemblage parallèle. Ce fait sera détaillé au chapitre 4.

3.1.1 Renumérotation et Factorisation Symbolique par Blocs

Les deux méthodes les plus utilisées à l'heure actuelle pour la renumérotation sont celles du degré minimum et des dissections emboîtées. L'idée de l'algorithme du degré minimum [68] est de choisir à chaque étape le pivot correspondant au sommet de plus petit degré du graphe associé à la matrice. L'algorithme de dissections emboîtées [28] consiste à déterminer un ensemble de sommets S qui sépare le graphe en deux parties A et B non voisines, numéroter S avec les plus grands numéros disponibles, et recommencer récursivement sur les sous-graphes A et B jusqu'à ce que leurs tailles deviennent inférie1ures à une valeur limite. Cette numérotation garantit qu'aucun terme non nul ne peut apparaître lors de la factorisation entre des inconnues de A et des inconnues de B. Dans les distributions logicielles qui font référence actuellement [12, 42, 46, 56, 57, 66], seuls les premiers niveaux de récursion de la dissection emboîtée sont effectués, les sous-graphes restants étant alors numérotés par degré minimum en dessous d'une certaine taille.

La renumérotation utilisée et basée sur le logiciel SCOTCH de partitionnement de graphes développé par Francois Pellegrini, correspond à un bon compromis entre la conservation du creux et l'équilibrage de l'arbre d'élimination. ORDER utilise ainsi la méthode de dissections emboîtées jusqu'à un certain seuil qui correspond à un nombre de sommets de l'ordre de quelques centaines dans la plupart des cas, puis les sous-graphes sont renumérotés en utilisant l'algorithme du "Halo Approximate Minimum Degree" [57] à partir de ce seuil.

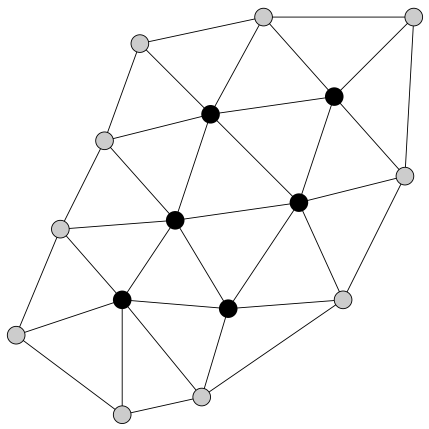

FIG. 3.3: Sans Halo.

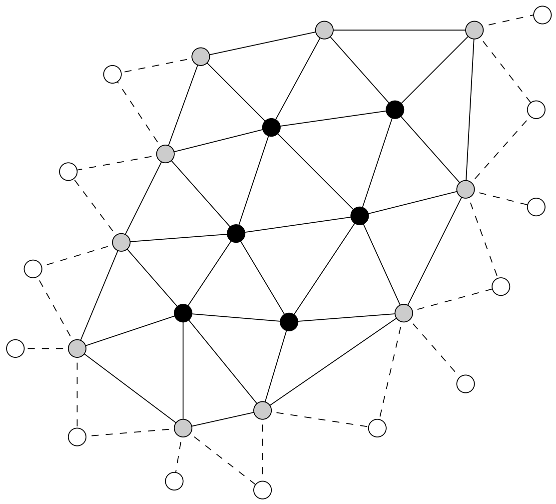

FIG. 3.4: Avec Halo.

Cette approche consiste à ajouter aux sous-graphes passés à l'algorithme de degré minimum une description topologique de leur voisinage immédiat (voir figures 3.3 et 3.4). Cette information supplémentaire, appelée *Halo*, est alors exploitée par l'algorithme de degré minimum approché ainsi modifié (degré minimum sous contrainte) pour numéroter les sommets de ces sous-graphes restants.

En complément du calcul de la renumérotation des inconnues, cet algorithme construit une partition des colonnes en blocs colonnes. Cette partition est réalisée en fusionnant le partitionnement des séparateurs calculés par l'algorithme de dissections emboîtées et les super-variables amalgamées pour chaque sous-graphe renuméroté à l'aide du degré minimum approché en tenant compte des informations de *Halo*.

La factorisation symbolique de la matrice renumérotée se fait elle aussi par blocs, afin de réduire la taille des structures de données manipulées par l'algorithme, ainsi que le temps pris par celui-ci, puisqu'il a été montré que, lorsque la factorisation symbolique s'effectue par blocs, sa complexité en temps est quasi-linéaire par rapport au nombre de blocs (la linéarité étant prouvée dans le cas de la dissection complète [15]).
L'algorithme de factorisation symbolique opère sur la matrice renumérotée et partitionnée en blocs-colonnes par la phase de renumérotation, et produit en sortie une nouvelle matrice décomposée par blocs, plus dense, et permettant de stocker l'ensemble des termes de remplissage qui seront produits au cours de la phase de factorisation numérique.

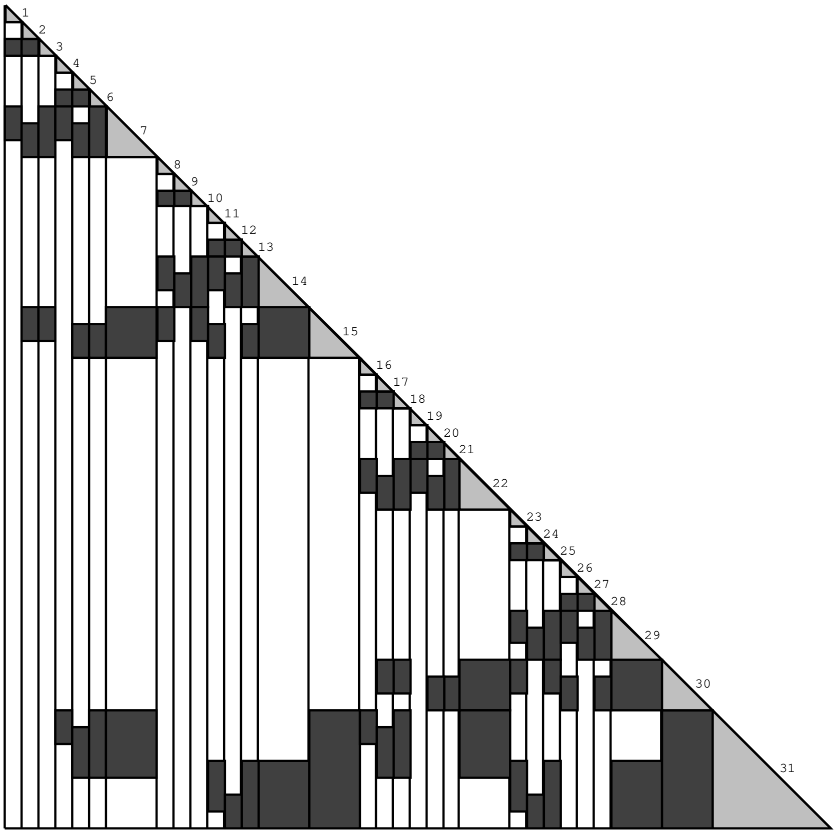

FIG. 3.5: Un exemple de matrice factorisée structurée par bloc colonne.

On consultera [57] pour une étude comparative réalisée sur des maillages réguliers et irréguliers industriels, montrant les performances très favorables du logiciel SCOTCH [56] comparées à celles du logiciel de référence METIS de Karypis et Kumar [46]. La grande majorité de ces matrices tests avec leurs caractéristiques est donnée à l'annexe A.

3.1.2 Repartitionnement et distribution des blocs

Certains solveurs numériques prennent entièrement à leur charge le séquencement dynamique des calculs de factorisation des blocs sur l'architecture parallèle, en se basant sur le placement des blocs préalablement calculé par un algorithme de prétraitement par exemple de type *subtree-to-subcube*.
L'approche qui a été choisie consiste à décharger entièrement le solveur de cette tâche, et donc à pré-calculer totalement l'ordonnancement des calculs de factorisation des blocs de manière statique, en simulant le plus précisément possible le comportement de l'architecture parallèle utilisée [35, 43, 44]. L'algorithme comprend

deux phases distinctes.

La première est la phase de repartitionnement, dans laquelle les plus gros blocs sont redécoupés afin de pouvoir faire participer plusieurs processeurs à leur traitement. On exhibe ainsi un parallélisme de type *calcul parallèle sur des blocs pleins*, qui s'applique principalement aux plus gros séparateurs de l'arbre d'élimination à la fin de la factorisation. Au cours de cette phase, on parcourt l'arbre d'élimination de haut en bas et on calcule pour chaque bloc un ensemble de processeurs candidats susceptibles de le traiter ; cette méthode est basée sur la technique de "proportional mapping" introduite par Pothen et Sun [58]. Cette notion de processeur candidat est indispensable pour préserver la bonne localité des communications ; celle-ci s'obtient en suivant la logique de l'arbre d'élimination, car ne seront candidats pour un nœud de l'arbre d'élimination que les processeurs ayant participé à la factorisation du sous-arbre dont il est la racine.

La deuxième phase est la phase de distribution proprement dite, dans laquelle on attribue à chaque bloc le processeur qui le calculera, parmi ses processeurs candidats. Pour ce faire, on remonte l'arbre d'élimination des feuilles vers la racine et, pour distribuer chaque nœud de l'arbre sur les processeurs, on calcule le temps (estimé) pris par sa factorisation et l'envoi de ses contributions. Ainsi, le choix de placement d'un bloc sur l'un de ses processeurs candidats est décidé en connaissant la date à laquelle potentiellement le processeur aura terminé de le traiter. Le processeur élu sera alors celui qui pourra avoir fini de traiter le bloc le plus tôt. Cette étape de simulation permet un ordonnancement fin des calculs et des communications ; elle nécessite le calibrage des opérations BLAS3 et des opérations de communication MPI pour l'architecture cible (homogène). Un certain nombre de modules permettent de réaliser préalablement ce calibrage.

Comme il a été constaté que des distributions hybrides (1D en bas de l'arbre d'élimination et 2D en haut) [44] donnent de meilleurs résultats, l'algorithme de repartitionnement a été modifié pour que le choix du type de distribution soit réalisé lors de la descente récursive de l'arbre d'élimination par blocs. Le critère de changement du type de distribution est actuellement un critère de niveau (ou de seuil) dans l'arbre d'élimination. Des travaux sont actuellement en cours pour améliorer ce choix, et en particulier pour exprimer ce critère en fonction de la taille des blocs colonnes et du nombre de processeurs.

L'annexe B donne pour notre jeu de cas tests les temps de cette phase de repartitionnement/distribution, montrant son faible coût en temps.

3.2 Enchaînement pour le traitement parallèle

Le traitement parallèle des données, détaillé à la figure 3.6, regroupe les modules :

- PABLAS : ce module réalise l'assemblage parallèle de la matrice de rigidité globale et/ou du second membre (voir chapitre 1). Il utilise les informations qui lui sont données par toutes les étapes séquentielles qui l'ont précédées : la distribution des éléments (elle même induite par la distribution des blocs de la matrice), la renumérotation des nœuds effectuée dans le module ORDER. PABLAS prend en entrée toutes les structures de données allouées par les modules du pré-traitement séquentiel, alloue les coefficients de la matrice étendue en termes de Degrés de Liberté, et réalise numériquement l'assemblage. Une description complète de cet assemblage parallèle est donnée dans le chapitre 5.
- SOPALIN : ce module représente le solveur parallèle de la chaîne logicielle EMILIO. Ce solveur réalise une factorisation Crout LDL^T sur une distribution 1D ou 1D/2D. C'est un solveur de type supernodal avec aggrégation totale (faite localement) de l'ensemble des contributions des blocs locaux aux blocs non locaux. Ainsi, on limite au strict minimum les communications inter-processeurs. Dans une deuxième étape, le module SOPALIN effectue une opération de "descente-remontée" parallèle, utilisant la même distribution que pour la factorisation, pour calculer un vecteur solution étant donné un vecteur second membre. Cette étape peut être appelée spécifiquement pour enchaîner des étapes de résolution sans avoir à refactoriser la matrice. SOPALIN utilise les structures de données qui lui sont données essentiellement par les modules BLEND et PABLAS et il modifie uniquement les coefficients de la matrice, puisque la structure de la matrice est la même avant et après la factorisation.

  La méthodologie algorithmique qui est à la base de ce module SOPALIN (travaux de Pierre Ramet en collaboration avec Pascal Hénon et Jean Roman [43, 44]) est présentée dans le paragraphe 3.2.1.
- ELAS : ce module est un peu "particulier" ; en effet, c'est le module du code OSSAU (écrit en Fortran) qui est chargé de calculer les matrices et les seconds membres élémentaires. Il est indépendant de la bibliothèque de fonctions EMILIO. Ce module utilise les structures de données allouées en statique par le code Fortran et il renvoie au module PABLAS un vecteur qui contient soit la matrice élémentaire, soit le second membre élémentaire calculé. Ce module est séquentiel et la mémoire nécessaire pour son exécution est précisée dans le chapitre 6.

Les phases de "mise aux conditions limites" et de "calcul du résidu" sont assez simples et ne posent pas de problème au niveau de la parallélisation.

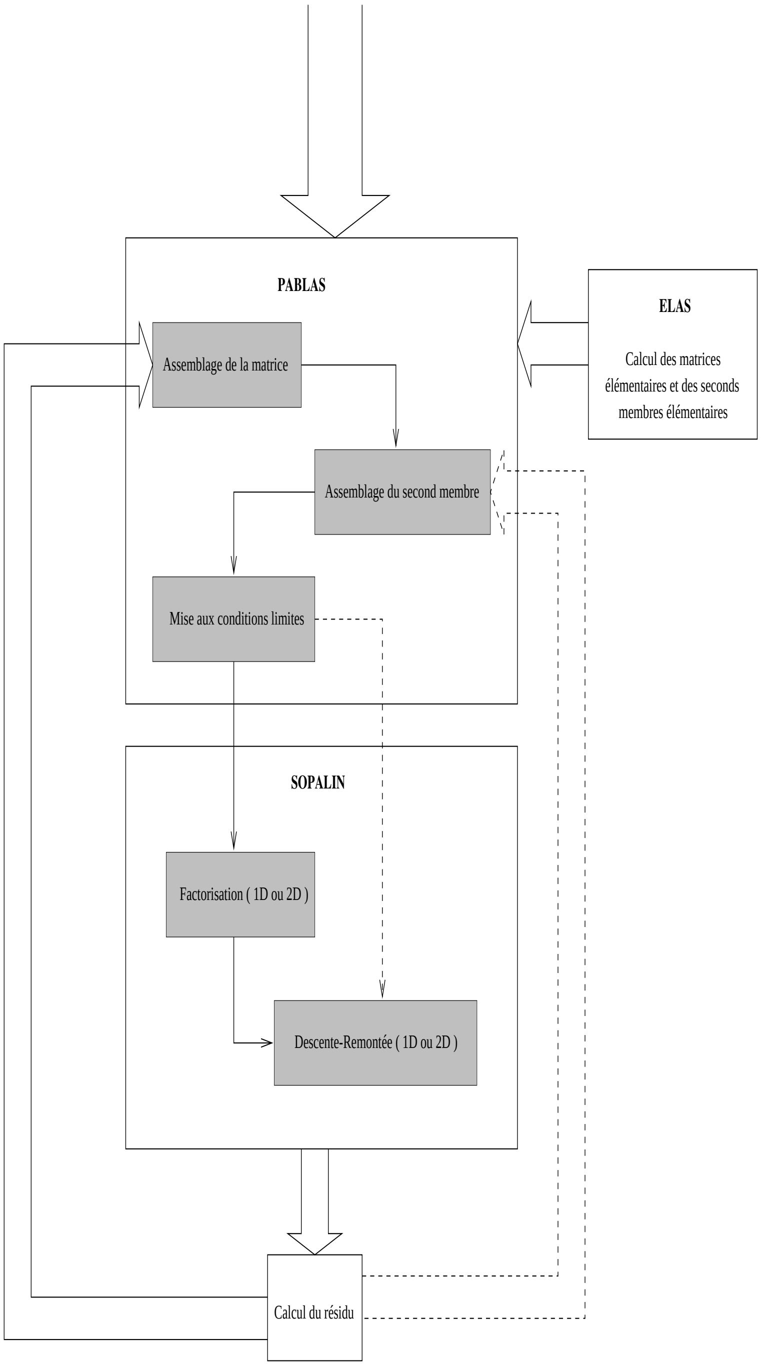

Fig. 3.6: Les étapes parallèles du projet EMILIO.

3.2.1 Le solveur parallèle

Comme on l'a dit précédemment, la phase de résolution numérique réalise une factorisation parallèle de Crout LDL^T avec une distribution 1D des blocs de la matrice (c'est-à-dire par bloc colonne) et/ou 2D (le bloc diagonal et les blocs extra-diagonaux d'un bloc-colonne ne seront pas forcément tous distribués sur le même processeur). Dans la première partie de ce paragraphe, nous présenterons le solveur qui est à l'heure actuelle utilisé dans EMILIO et ce solveur est basé sur une distribution 1D des blocs de la matrice [43]. Dans la seconde partie, nous présenterons l'extension de ce solveur à une distribution 2D des blocs [44].

On considère la structure de données par blocs de la matrice factorisée L calculée par la factorisation symbolique par blocs. N est le nombre de blocs colonnes. Nous définissons la fonction booléenne $\mathit{off\_diag}(k, j)$, $1 \leq k < j \leq N$ qui retourne *vrai* si et seulement s'il existe un bloc extra-diagonal dans le bloc-colonne k "en face" du bloc-colonne j. On peut alors construire les deux ensembles :

$$\begin{cases} BStruct(L_{k*}) & := & \{i < k \mid \mathit{off\_diag}(i,k) = \mathit{vrai}\} \\ BStruct(L_{*k}) & := & \{j > k \mid \mathit{off\_diag}(k,j) = \mathit{vrai}\}. \end{cases}$$

Ainsi, $BStruct(L_{k*})$ est l'ensemble des blocs-colonnes qui modifient le bloc-colonne k et $BStruct(L_{*k})$ est l'ensemble des blocs colonnes qui sont modifiés par le bloc-colonne k (voir figure 3.7).

On s'intéresse maintenant à l'algorithme supernodal de la factorisation parallèle creuse LDL^t où toutes les contributions distantes sont localement agrégées dans une structure bloc. Ce schéma est proche de l'algorithme fan-in [7] ; chaque processeur communique uniquement des blocs-colonnes de contributions agrégées. Ces structures, que nous noterons par la suite $AUCB_k$ (*aggregate update column block*), peuvent être facilement construites grâce à l'étape de factorisation symbolique. Il est important de remarquer qu'un bloc-colonne k ne recevra que des contributions de la part des processeurs dans l'ensemble $Procs(L_{k*}) := \{map(i) \mid i \in BStruct(L_{k*})\}$, où l'opérateur $map()$ correspond à la distribution des blocs-colonnes.

Le "pseudo-code" de la factorisation LDL^t peut-être exprimé en terme de calculs par blocs (voir figure 3.8) ; ces calculs sont réalisés, autant que possible, sur des ensembles de blocs compactés pour profiter au mieux des effets BLAS.

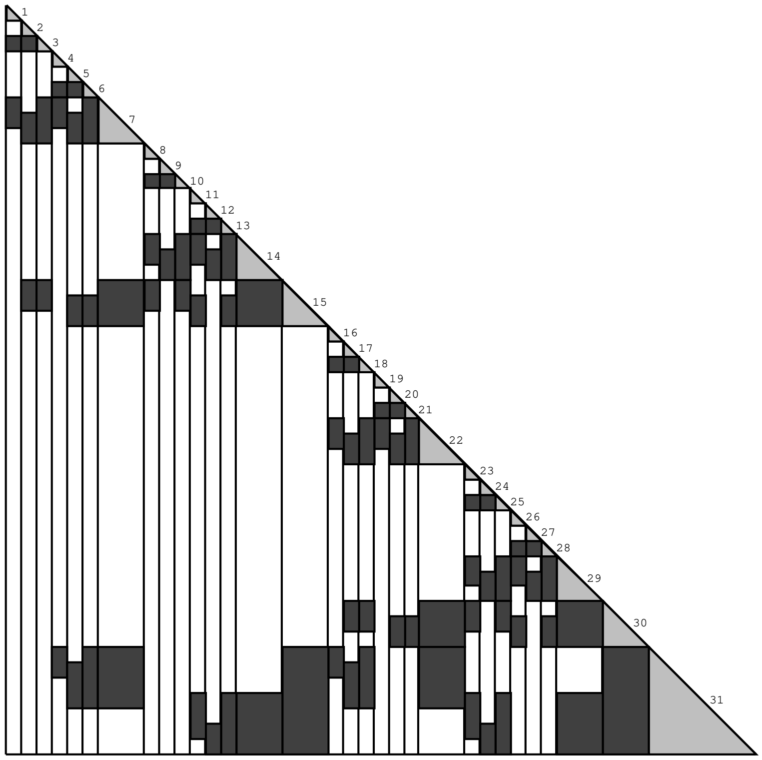

FIG. 3.7: Un exemple de matrice factorisée structurée par bloc-colonne. On a $BStruct(L_{7*}) := \{1, 2, 3, 4, 5, 6\}$ et $BStruct(L_{*7}) := \{15, 31\}$.

Les synchronisations de l'algorithme sont assurées par l'utilisation des variables r_k qui comptent le nombre de contributions restant à prendre en compte pour le bloc-colonne k. Tant que $r_k \neq 0$, il reste des contributions à recevoir pour le bloc-colonne k (lignes 2–4).

Ensuite le bloc diagonal peut-être factorisé et les blocs extra-diagonaux sont mis à jour (lignes 6–8). Le calcul partiel (F) est sauvegardé pour être réutilisé dans le calcul des contributions. Toutes les contributions associées au bloc-colonne k peuvent maintenant être calculées et ajoutées localement (ligne 12) ou agrégées (ligne 14).

Un bloc-colonne de contributions agrégées peut-être envoyé si toutes ses contributions locales ont été prises en compte (ligne 15). Pour limiter l'augmentation mémoire due au stockage des aggrégations locales, ces structures sont allouées dynamiquement seulement et sont désallouées dès que leur émission est achevée. Dans le cas où la ressource mémoire est critique, il est également possible d'envoyer ces structures avec une agrégation partiellement achevée afin de libérer de l'espace mémoire, ce qui s'apparente à un schéma de type fan-both [5].

Remarquons que la complexité de chacune de ces étapes dépend uniquement de la taille des blocs et peut donc être évaluée précisement.

Notations pour k fixé, $1 \leq k \leq N$:

- r_k : nombre de contributions restant à soustraire au bloc colonne k,
- r_k^o : valeur initiale de r_k,
- s_k : nombre de contributions restant à ajouter à $AUCB_k$,
- s_k^o : valeur initiale de s_k,
- le symbole $*$ signifie $\forall j \in BStruct(L_{*k})$,
- soit $j \geq k$; la séquence $[j]$ signifie $\forall i \in BStruct(L_{*k}) \cup \{k\}$ avec $i \geq j$.

Pour le processeur p :

For $k \in [1, N]$ tel que $map(k) == p$ **Do**
    While $r_k \neq 0$ **Do** % étape de réception/mise à jour
        Recevoir $AUCB_j$;
        $A_{[j]j} = A_{[j]j} - AUCB_j$; $r_j \leftarrow r_j - s_j^o$;
    % étape de calcul/émission
    Factoriser A_{kk} sous la forme $L_{kk}D_kL_{kk}^t$;
    Résoudre $L_{kk}F_*^t = A_{*k}^t$;
    Résoudre $D_kL_{*k}^t = F_*^t$;
    For $j \in BStruct(L_{*k})$ **Do**
        Calculer $C = L_{[j]k}F_j^t$;
        If $map(j) == p$ **Then**
            $A_{[j]j} = A_{[j]j} - C$; $r_j \leftarrow r_j - 1$;
        Else % étape d'agrégation
            $AUCB_j = AUCB_j + C$; $s_j \leftarrow s_j - 1$;
            If $s_j = 0$ **Then** envoyer $AUCB_j$ à $map(j)$;

FIG. 3.8: Algorithme de la factorisation parallèle pour une distribution 1D.

Dans notre implémentation du solveur, les calculs par blocs sont ordonnés de façon à respecter le rang avec lequel le bloc-colonne à été distribué. Pour un processeur p, on associe donc le vecteur K_p contenant les N_p blocs-colonnes locaux et ordonnés par leur priorité. Mais il faut aussi, pour que le solveur respecte les prédictions de l'étape de distribution, ordonner de façon compatible les émissions des blocs-colonnes de contributions agrégées.

L'algorithme de la factorisation parallèle devra donc être modifié pour prendre en compte ces extensions (voir figure 3.9). Les émissions sont réalisées à l'aide de l'ensemble $AUCB_{ready}$ contenant l'ensemble des blocs-colonnes de contributions agrégées prêts à être envoyés au processeur $map(j)$ et dont les priorités sont plus élevées que le premier bloc-colonne de contributions agrégées non prêt.

Pour le processeur p :
1. **For** $i = 1$ **To** N_p **Do**
   $k = K_p(i)$
   [...]
3. recevoir $AUCB_k$;
4. $A_{[k]k} = A_{[k]k} - AUCB_k$; $r_k \leftarrow r_k - s_k^o$;
   [...]
15. **If** $s_j = 0$ **Then** envoyer $AUCB_{ready}$ à $map(j)$;

FIG. 3.9: Modifications apportées à l'algorithme de factorisation 1D.

On voit donc l'intérêt de l'étape de pré-traitement réalisée par le module BLEND (cf. paragraphe 3.1.2). Le solveur va uniquement et exactement fonctionner en respectant l'ordonnancement des calculs et des communications prédit par la simulation. Celle-ci étant couplée avec la distribution, on est en mesure aussi *de tenir compte des effets d'aggrégation locale en mémoire.*

En distribution 2D des blocs de la matrice, on s'intéresse toujours à l'algorithme supernodal de la factorisation parallèle creuse LDL^T où toutes les contributions distantes sont agrégées localement dans une structure bloc. Un bloc j dans un bloc-colonne k recevra donc un bloc de contributions agrégées, noté AUB_{jk} par la suite, seulement de la part des processeurs dans l'ensemble
$Procs(L_{jk}) := \{map(j,i) \mid i \in BStruct(L_{k*})\ and\ j \in BStruct(L_{*i})\}$, où l'opérateur $map(,)$ correspond à la distribution 2D des blocs.

Notations pour k fixé, $1 \leq k \leq N$:

- N_p : nombre total de tâches pour le processeur p,
- $K_p[n]$: $n^{\text{ième}}$ tâche locale pour le processeur p,
- pour un bloc-colonne k, le symbole $\star$ signifie $\forall j \in BStruct(L_{*k})$,
- soit $j \geq k$; la séquence $[j]$ signifie $\forall i \in BStruct(L_{*k}) \cup \{k\}$ avec $i \geq j$,
- $map([j], j)$ représente, par extension, $map(i, j)$ pour chaque valeur de $i \in [j]$.

Pour le processeur p :

For $n = 1$ to N_p **Do**
  Switch (Type($K_p[n]$)) **Do**
    `COMP1D(k)` : Recevoir tous les $AUB_{[k]k}$ pour $A_{[k]k}$
      Factoriser A_{kk} sous la forme $L_{kk}D_kL_{kk}^t$
      Résoudre $L_{kk}F^t = A_{\star k}^t$ et $D_kL_{\star k}^t = F^t$
      For $j \in BStruct(L_{*k})$ **Do**
        Calculer $C_{[j]} = L_{[j]k}F_j^t$
        If $map([j], j) == p$ **Then** $A_{[j]j} = A_{[j]j} - C_{[j]}$
        Else $AUB_{[j]j} = AUB_{[j]j} + C_{[j]}$, si prêt envoyer $AUB_{[j]j}$ à $map([j], j)$
    `FACTOR(k)` : Recevoir tous les AUB_{kk} pour A_{kk}
      Factoriser A_{kk} sous la forme $L_{kk}D_kL_{kk}^t$ et envoyer $L_{kk}D_k$ à $map([k], k)$
    `BDIV(j,k)` : Recevoir $L_{kk}D_k$ et tous les AUB_{jk} pour A_{jk}
      Résoudre $L_{kk}F_j^t = A_{jk}^t$ et $D_kL_{jk}^t = F_j^t$ et envoyer F_j^t à $map([j], k)$
    `BMOD(i,j,k)` : Recevoir F_j^t
      Calculer $C_i = L_{ik}F_j^t$
      If $map(i, j) == p$ **Then** $A_{ij} = A_{ij} - C_i$
      Else $AUB_{ij} = AUB_{ij} + C_i$, si prêt envoyer AUB_{ij} à $map(i, j)$

FIG. 3.10: Algorithme de la factorisation parallèle pour une distribution mixte 1D/2D.

Le “pseudo-code” de la factorisation LDL^t peut-être exprimé en terme de calculs par blocs (voir figure 3.10) ; les calculs effectifs sont réalisés, autant que possible, sur des ensembles de blocs compactés pour profiter au mieux des effets BLAS.

L’algorithme proposé permet de prendre en compte à la fois une distribution 1D et 2D ; un bloc-colonne distribué en 1D est vu comme s’il était distribué en 2D avec tous les blocs extra-diagonaux placés sur le même processeur ; cependant, ce cas particulier est traité spécifiquement au niveau du code pour effectuer les calculs sur l’ensemble des blocs compactés. Les calculs par blocs, considérés comme des *tâches*, sont totalement ordonnés lors de l’étape de distribution. Les émissions des blocs de contributions agrégées sont également ordonnées de façon compatible. On peut alors définir les quatre types de tâche qui apparaissent à la figure 3.10 :

- `COMP1D(k)` : mise à jour et calcul des contributions pour le bloc colonne k dans le cas d'une distribution 1D,
- `FACTOR(k)` : factorisation du bloc diagonal k,
- `BDIV(j,k)` : mise à jour du bloc extra-diagonal j dans le bloc-colonne k avec $k < j \leq N$,
- `BMOD(i,j,k)` : calcul de la contribution du bloc i dans le bloc-colonne k “en face” du bloc-colonne j avec $k < j \leq i \leq N$.

Les annexes C, D, E, F et G donnent respectivement pour nos cas tests :

- les temps d'exécution de la factorisation parallèle pour une distribution 1D,
- des diagrammes comparant les prédictions de la simulation et le comportement à l'exécution du solveur pour un cas régulier et deux cas irréguliers,
- les temps d'exécution de la descente-remontée parallèle pour une distribution 1D,
- les temps d'exécution de la factorisation parallèle pour une distribution 2D,
- et une comparaison entre notre solveur SOPALIN utilisant la numérotation issue de ORDER, et PSPASES qui est souvent considéré comme un solveur de référence pour ce type de système [45] ; ce dernier utilise la numérotation issue de METIS et une approche multifrontale.

Chapitre 4

Distribution des éléments

4.1 Introduction

Dans le chapitre 1 nous avons vu que l'assemblage consistait à calculer les matrices élémentaires des éléments du maillage et "additionner" ces matrices à la matrice globale de rigidité K. On rappelle qu'une matrice élémentaire décrit les propriétés physiques d'un élément et la complexité de son calcul dépend essentiellement du type d'élément fini et des contraintes dans le cas non linéaire (voir chapitre 1). Puisque la matrice globale de rigidité est distribuée (voir chapitre 3), le calcul et l'addition de matrices élémentaires dans les parties locales de la matrice globale K présente deux problèmes majeurs :

- équilibrer les coûts de calcul des matrices élémentaires sur l'ensemble des processeurs,
- minimiser le coût des communications induites par les mises à jour de la matrice K.

Nous allons présenter notre algorithme de distribution des éléments qui essaie de résoudre ces deux problèmes ainsi que des exemples de distribution pour des cas tests réels provenant du CEA/CESTA.

4.2 Algorithmique et mise en œuvre

Dans le chapitre 3, nous avons vu que la phase de renumérotation de notre prétraitement séquentiel utilise un couplage "fin" de l'algorithme de degré minimum et de l'algorithme de dissections emboîtées. Le *partitionnement du graphe d'éléments finis* (associé à la matrice K) en super-variables est obtenu en fusionnant la partition, constituée des séparateurs (issus de la dissection emboîtée) avec la partition des super-variables amalgamées localement dans chaque sous-graphe (numéroté par l'algorithme Halo-AMD). Cette décomposition topologique du graphe d'éléments finis implique une décomposition "naturelle" de l'ensemble des éléments du maillage en sous-domaines indépendants. Les inconnues portées par des nœuds situés dans des sous-graphes indépendants et éliminés en parallèle lors de la factorisation, appartiennent à des éléments localisés dans des sous-domaines indépendants du maillage. L'assemblage de ces éléments peut donc s'effectuer en parallèle.

Pour effectuer la distribution des éléments sur les processeurs, il est nécessaire de connaître non seulement la distribution des blocs, mais aussi des informations relatives au maillage : géométrie de chaque élément fini, numéros des nœuds composant cet élément, et nombre de Degrés De Liberté par nœud. Toutes ces informations sont fournies lors de la lecture du maillage de la structure, ce maillage étant dupliqué au niveau de chaque processeur. Il est important de noter que les numéros des nœuds composant les éléments finis ont été renumérotés par le module SCOTCH, mais que l'on a décidé de conserver la numérotation des éléments. Afin d'optimiser les accès à la structure décrivant la matrice, on effectue pour chaque élément fini un tri des numéros de nœuds le composant. On va donc manipuler un maillage modifié par rapport au maillage issu du numéroteur initial, mais ce dernier pourra être récupéré grâce à des tableaux de permutations permettant de passer de la nouvelle numérotation à l'ancienne et vice versa.

Notre algorithme équilibre la masse de calculs au sens où chaque processeur devra calculer *un nombre équivalent de matrices élémentaires*. D'autre part, les communications induites par les éléments frontières ne sont pas prises en compte pour l'équilibrage car *elles sont supposées recouvertes par les calculs* [37].

Les éléments sont composés de nœuds correspondants à des colonnes de la matrice qui est partitionée et distribuée par le module BLEND. Ce dernier prend en compte les coûts de calcul associés aux nœuds, et ces coûts sont inégaux suivant la connexité du nœud dans le maillage. BLEND peut donc attribuer un nombre de colonnes (donc de nœuds) très différent d'un processeur à l'autre, ce qui peut entraîner un déséquilibre au niveau de l'assemblage. Le module SACHA dont le rôle va être de distribuer les éléments se décompose alors en deux parties principales :

- Dans la première phase, on construit pour chaque élément la liste des processeurs qui possèdent un ou plusieurs nœuds de cet élément (et donc les colonnes associées à ces nœuds), ces nœuds ayant été attribués par la distribution des blocs colonnes. Si cette liste se réduit à un seul processeur P, l'élément est *totalement local* à P, sinon cet élément est *non totalement local*. Grâce à cette boucle sur les éléments du maillage, on calcule le nombre $M(P)$ d'éléments totalement locaux pour chaque processeur P, puis on détermine M le minimum des $M(P)$ et on attribue M éléments totalement locaux à chaque processeur. Nous stockons ensuite dans un tableau T les éléments restants (c.a.d. les éléments totalement locaux qui n'ont pas pu être attribués lors de la première boucle et l'ensemble des éléments non totalement locaux) avec pour chaque élément le nombre de processeurs propriétaires.

– Dans la deuxième phase, on trie d'abord le tableau T suivant l'ordre *décroissant* du nombre de processeurs propriétaires de nœuds. Puis, on va déterminer sur quel processeur chaque élément doit être placé, en favorisant d'abord le processeur qui possède le plus grand nombre de nœuds de cet élément (afin d'obtenir un critère de localité pour les ajouts de contributions), et ensuite le nombre d'éléments déja affectés à ce processeur (il s'agit d'un algorithme glouton multicritères hiérarchisés). Enfin, pour les éléments totalement locaux qui n'ont pas pu être attribués lors de la première phase, on cherche systématiquement à minimiser la distance, en terme de numéros logiques de processeurs, entre le processeur possédant les nœuds de l'élément et le processeur sur lequel celui-ci sera placé, afin d'augmenter encore la localité des communications.

Le module SACHA détermine pour chaque processeur la masse de calculs à effectuer, mais aussi la quantité de communications à recevoir et à envoyer ; ces informations permettront de “piloter” un assemblage parallèle efficace. Pour illustrer notre stratégie nous présentons (figure 4.1) une grille 5×5 renumérotée par un algorithme de type dissections emboîtées et l'arbre d'élimination correspondant à cette grille renumérotée (figure 4.2).

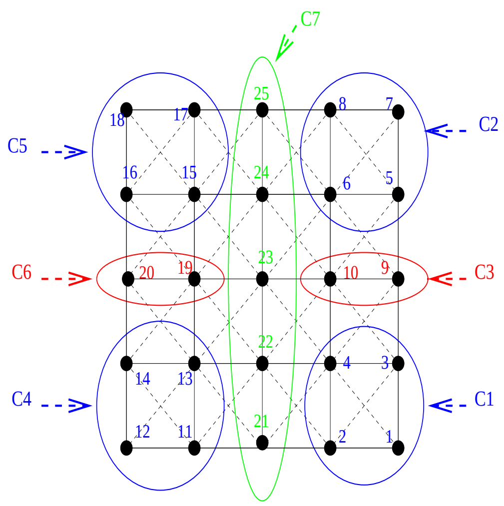

Fig. 4.1: Un exemple de dissections sur une grille 5×5.

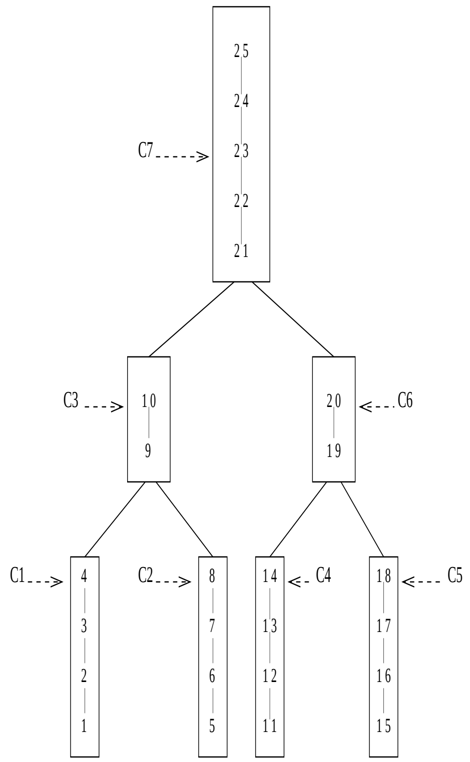

FIG. 4.2: L'arbre d'élimination correspondant à la grille 5×5

Pour l'exemple de cette grille 5×5, nous décidons de numéroter les éléments du maillage de gauche à droite et de haut en bas. On obtient donc 16 éléments avec 4 nœuds chacun. La décomposition en éléments de la grille 5×5 en éléments à 4 nœuds après renumérotation est donc la suivante :

- l'élément numéro 1 possède les nœuds 18 17 15 16,
- l'élément numéro 2 possède les nœuds 17 25 24 15,
- l'élément numéro 3 possède les nœuds 25 8 6 24,
- l'élément numéro 4 possède les nœuds 8 7 5 6,
- l'élément numéro 5 possède les nœuds 16 15 19 20,
- l'élément numéro 6 possède les nœuds 15 24 23 19,
- l'élément numéro 7 possède les nœuds 24 6 10 23,
- l'élément numéro 8 possède les nœuds 6 5 9 10,
- l'élément numéro 9 possède les nœuds 20 19 13 14,
- l'élément numéro 10 possède les nœuds 19 23 22 13,
- l'élément numéro 11 possède les nœuds 23 10 4 22,

- l'élément numéro 12 possède les nœuds 10 9 3 4,
- l'élément numéro 13 possède les nœuds 14 13 11 12,
- l'élément numéro 14 possède les nœuds 13 22 21 11,
- l'élément numéro 15 possède les nœuds 22 4 2 21,
- l'élément numéro 16 possède les nœuds 4 3 1 2.

On constate sur les figures 4.1 et 4.2 que les éléments 1 4 13 16 ont chacun leurs nœuds situés dans le même bloc-colonne, ce qui permet d'affirmer que ces éléments seront des éléments totalement locaux. Les autres éléments du maillage seront des éléments totalement locaux ou non totalement locaux, ceci dépendant du nombre de processeurs et du seuil de découpage des blocs indiqué dans la phase de partitionnement et de distribution des blocs. En observant l'arbre d'élimination de la figure 4.2, on remarque que les nœuds des éléments 1 $(18, 17, 15, 16)$, 4 $(8, 7, 5, 6)$, 13 $(14, 13, 12, 11)$, 16 $(4, 3, 1, 2)$ correspondent aux feuilles de cet arbre d'élimination et que notre stratégie de distribution des éléments non totalement locaux, *basé sur un ordre décroissant du nombre de processeurs propriétaires (tableau T)*, correspond à un cheminement *de haut en bas dans cet arbre.*

Nous donnons pour terminer ce paragraphe divers schémas représentant un exemple de maillage : le cas test **COLOGB15** qui sera détaillé dans le chapitre 7. Ces 3 schémas correspondent respectivement au graphe des éléments du maillage, au graphe des nœuds et enfin au partitionnement induit par l'algorithme de dissections emboîtées appliqué par le logiciel SCOTCH.

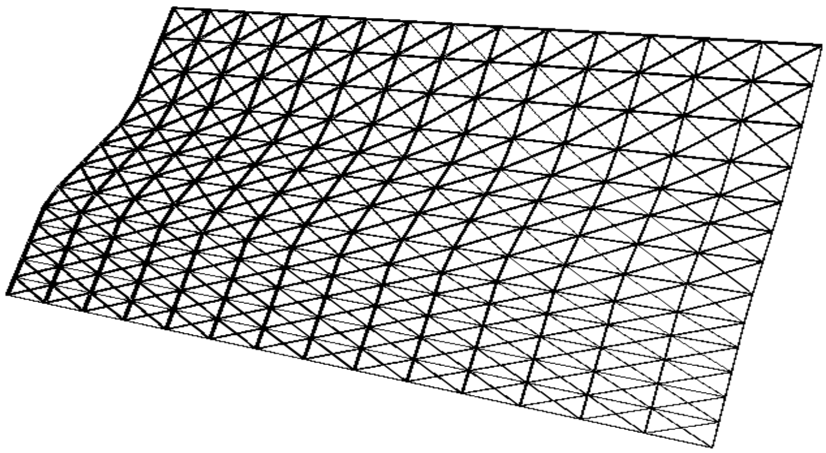

FIG. 4.3: Le graphe des éléments de COLOGB15.

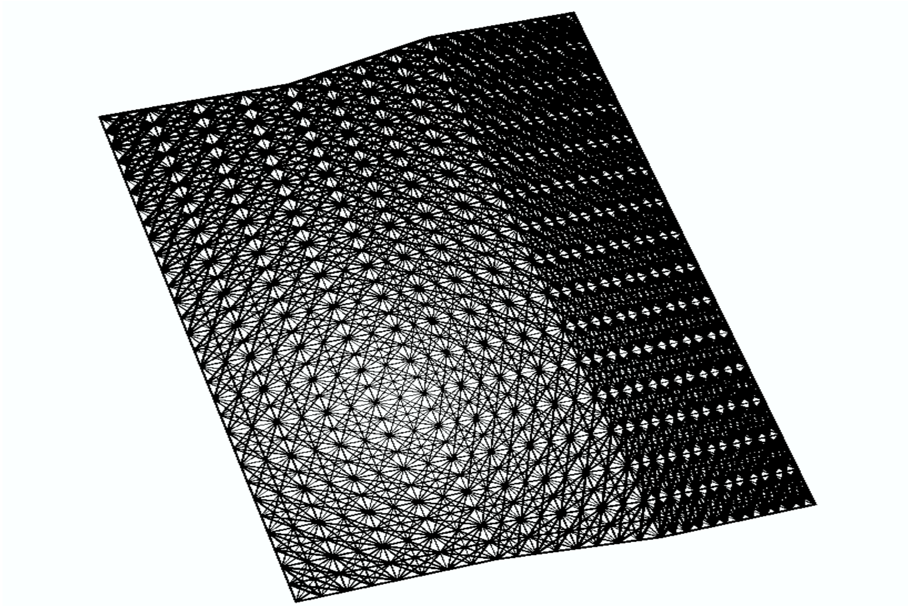

FIG. 4.4: Le graphe des nœuds pour COLOGB15.

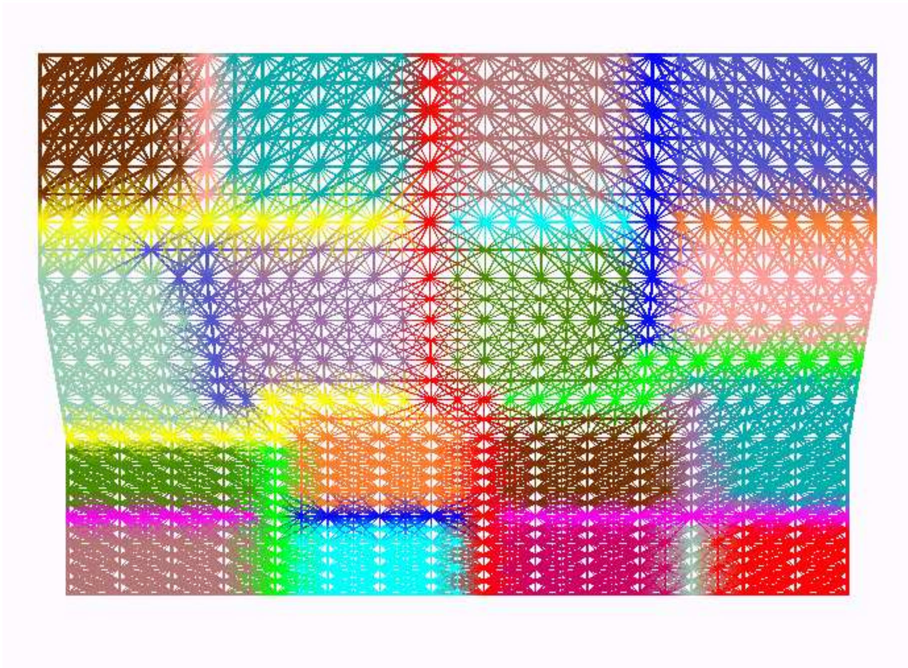

FIG. 4.5: Dissections emboîtées (4 niveaux de récursion) obtenues par SCOTCH sur COLOGB15.

4.3 Validation expérimentale de la distribution des éléments finis

Dans cette partie, nous donnons des résultats pour la distribution des éléments faite par le module SACHA pour une collection de cas tests réguliers et non réguliers. Pour chaque processeur nous donnons les statistiques suivantes :

- le nombre d'éléments totalement locaux,
- le nombre d'éléments non totalement locaux,
- le nombre total d'éléments attribués,
- le nombre de colonnes pour des contributions extérieures (par "extérieur", on entend des colonnes n'appartenant pas à des blocs locaux au processeur).

De plus, nous donnerons l'écart maximum (en pourcentage) entre les nombres maximum et minimum d'éléments attribués à un processeur. Cette valeur permettra de vérifier que notre algorithme de distribution équilibre bien la masse de calculs entre les différents processeurs.

Nous présentons trois cas tests avec un nombre de processeurs égal à 64. On a choisi de ne présenter la distribution qu'avec 64 processeurs pour bien illustrer la répartition des éléments sur un grand nombre de processeurs. Ces trois cas tests proviennent de maillages du CEA (décrits dans les chapitres 7 et 8). Pour chacun des cas tests, nous donnons les caractéristiques des maillages, c'est-à-dire le nombre d'éléments et le nombre de nœuds. Nous indiquerons les temps de distribution dans les chapitres 7 et 8.

Légende pour les tableaux :

- p : numéro du processeur,
- loc : nombre d'éléments totalement locaux attribués au processeur,
- nloc : nombre d'éléments non totalement locaux attribués au processeur,
- tot : nombre total d'éléments attribués au processeur,
- ext : nombre de colonnes pour des contributions extérieures.

4.3.1 COLOGB75 avec 64 processeurs

Nombre total d'éléments = 3150
Nombre total de nœuds = 16736

p	loc	nloc	tot	ext	p	loc	nloc	tot	ext
0	7	43	50	541	**32**	2	47	49	604
1	10	40	50	425	**33**	2	50	50	660
2	12	38	50	516	**34**	2	48	50	599
3	14	36	50	423	**35**	18	31	49	432
4	18	32	50	383	**36**	18	31	49	390
5	12	38	50	461	**37**	2	48	50	533
6	2	48	50	584	**38**	2	48	50	549
7	18	32	50	450	**39**	16	34	50	411
8	2	46	48	730	**40**	24	26	50	271
9	2	48	48	580	**41**	14	35	49	428
10	2	46	48	497	**42**	15	34	49	412
11	7	43	50	693	**43**	14	35	49	480
12	2	48	50	535	**44**	2	47	49	634
13	2	47	49	417	**45**	36	14	50	130
14	2	48	50	630	**46**	2	50	50	647
15	5	45	50	594	**47**	2	48	50	567
16	7	43	50	628	**48**	15	34	49	514
17	2	46	48	541	**49**	7	42	49	725
18	2	46	48	566	**50**	2	47	49	539
19	2	46	48	563	**51**	12	37	49	656
20	2	46	48	581	**52**	2	48	50	546
21	2	48	48	689	**53**	2	47	49	546
22	2	46	48	586	**54**	7	43	50	717
23	2	46	48	648	**55**	2	47	49	600
24	2	48	48	580	**56**	2	47	49	554
25	2	46	48	508	**57**	2	47	49	627
26	18	32	50	584	**58**	2	47	49	624
27	2	46	48	553	**59**	18	31	49	620
28	4	45	49	690	**60**	2	47	49	641
29	2	47	49	600	**61**	2	47	49	509
30	18	32	50	580	**62**	2	47	49	553
31	2	48	50	643	**63**	21	29	50	580

Ecart maximum : ((50 - 48) / 3150) * 100 = 0.0634 %

4.3.2 CEL3DM avec 64 processeurs

Nombre total d'éléments = 14400
Nombre total de nœuds = 10360

p	loc	nloc	tot	ext	p	loc	nloc	tot	ext
0	31	195	226	748	**32**	2	222	224	1097
1	35	191	226	752	**33**	2	222	224	1067
2	35	191	226	709	**34**	2	222	224	1100
3	60	166	226	645	**35**	2	222	224	1159
4	2	224	226	787	**36**	2	222	224	1180
5	57	169	226	618	**37**	2	223	225	1137
6	30	194	224	848	**38**	2	223	225	1129
7	37	187	224	844	**39**	2	223	225	1017
8	47	179	226	789	**40**	2	223	225	1054
9	50	176	226	804	**41**	2	223	225	1156
10	20	206	226	927	**42**	2	223	225	1145
11	14	212	226	989	**43**	2	223	225	1142
12	2	222	224	817	**44**	2	223	225	1110
13	107	119	226	679	**45**	2	223	225	1161
14	2	222	224	1038	**46**	2	223	225	1168
15	2	222	224	1083	**47**	2	223	225	1060
16	2	222	224	1069	**48**	2	224	226	709
17	2	222	224	1146	**49**	54	172	226	590
18	2	222	224	1130	**50**	55	171	226	513
19	2	222	224	1132	**51**	136	90	226	297
20	2	222	224	1158	**52**	73	153	226	460
21	2	222	224	1156	**53**	73	153	226	480
22	2	222	224	1141	**54**	91	135	226	397
23	2	222	224	1124	**55**	68	157	225	540
24	2	222	224	1113	**56**	79	147	226	486
25	2	222	224	1111	**57**	80	146	226	468
26	2	222	224	1084	**58**	82	144	226	483
27	2	222	224	1134	**59**	7	218	225	771
28	2	222	224	1114	**60**	92	134	226	488
29	2	222	224	1086	**61**	84	141	225	665
30	2	222	224	842	**62**	62	164	226	729
31	10	216	226	1310	**63**	68	158	226	727

Ecart maximum : ((226 - 224) / 14400) * 100 = 0.0138 %

4.3.3 COUPOLE3000T avec 64 processeurs

Nombre total d'éléments = 126000
Nombre total de nœuds = 663161

p	loc	nloc	tot	ext	p	loc	nloc	tot	ext
0	1859	109	1968	1984	**32**	1701	268	1969	4434
1	1548	421	1969	4943	**33**	1688	281	1969	3925
2	1842	127	1969	2430	**34**	1668	301	1969	4485
3	1559	409	1968	5659	**35**	1697	272	1969	4191
4	1548	420	1968	5598	**36**	1548	421	1969	5123
5	1681	287	1968	4593	**37**	1616	353	1969	6824
6	1735	233	1968	3183	**38**	1553	416	1969	5871
7	1858	111	1969	893	**39**	1594	375	1969	4639
8	1619	349	1968	4936	**40**	1554	415	1969	5435
9	1625	343	1968	4799	**41**	1553	416	1969	5878
10	1735	233	1968	4020	**42**	1811	158	1969	3088
11	1835	133	1968	1990	**43**	1548	421	1969	5631
12	1598	370	1968	5753	**44**	1548	421	1969	4831
13	1629	339	1968	5391	**45**	1671	298	1969	4943
14	1733	235	1968	4501	**46**	1548	421	1969	4756
15	1584	384	1968	4893	**47**	1659	310	1969	4742
16	1589	379	1968	4590	**48**	1548	421	1969	4499
17	1762	207	1969	3943	**49**	1843	126	1969	1514
18	1548	420	1968	5696	**50**	1548	421	1969	6526
19	1548	420	1968	6932	**51**	1599	370	1969	5367
20	1894	75	1969	852	**52**	1596	373	1969	5146
21	1834	135	1969	2085	**53**	1586	383	1969	5157
22	1566	403	1969	4947	**54**	1582	387	1969	5508
23	1548	421	1969	5440	**55**	1550	419	1969	5941
24	1843	126	1969	2459	**56**	1835	134	1969	1442
25	1796	173	1969	2855	**57**	1712	257	1969	3526
26	1548	421	1969	4753	**58**	1808	161	1969	2420
27	1714	255	1969	4636	**59**	1787	182	1969	3581
28	1548	421	1969	5926	**60**	1612	357	1969	5330
29	1548	421	1969	6492	**61**	1635	334	1969	4937
30	1562	407	1969	5933	**62**	1568	401	1969	6018
31	1627	342	1969	4690	**63**	1606	363	1969	5407

Ecart maximum : ((1969 - 1968) / 126000) * 100 = 0.0008 %

4.4 Analyse des résultats

On peut constater sur ces différents tableaux que l'écart maximum est toujours inférieur à 1 %, ce qui démontre que notre algorithme équilibre bien le nombre d'éléments attribués à chaque processeur. De même, on constate qu'à nombre de processeurs fixe (ici 64), le pourcentage d'éléments totalement locaux augmente naturellement d'autant plus que le nombre total d'éléments est grand. On verra au chapitre suivant que compte tenue de la stratégie de traitement des éléments sur les processeurs, ceci permettra un recouvrement calcul-communication d'autant plus important.

Chapitre 5

Assemblage parallèle

5.1 Introduction

Le chapitre 4 a présenté un algorithme "efficace" de distribution séquentielle des éléments du maillage. Dans ce chapitre 5, nous allons détailler la stratégie de notre assemblage parallèle en sachant que cet assemblage est complètement guidé par le pré-traitement séquentiel des données. Cet assemblage est actuellement basé sur une distribution 1D des blocs de la matrice [37], son évolution dans un avenir proche sera de s'adapter à une distribution 2D des blocs.

5.2 Algorithmique et mise en œuvre

L'assemblage parallèle utilise les informations qui lui sont données par la distribution des blocs colonnes de la matrice et par la distribution des éléments du maillage. Cette dernière lui fournit non seulement la liste des éléments dont les matrices élémentaires vont être calculées par chaque processeur, mais aussi des informations relatives aux communications : les tailles des messages à envoyer, le nombre de messages à recevoir, etc...). Toutes ces informations vont permettre d'allouer le strict minimum d'espace mémoire, de minimiser le coût des communications induites par les éléments non totalement locaux, et de recouvrir ces communications.

Le calcul des matrices élémentaires se fait par un appel du code OSSAU original. On a considéré au début de notre étude que le temps de calcul d'une matrice élémentaire est équivalent pour chaque élément. Ceci sera vrai en particulier avant la phase de plastification. Après la phase de plastification, on a vu dans la description du problème mathématique que le coût de calcul des matrices de rigidité de certains éléments pourra être plus important, et on verra dans la partie suivante "Validation

Expérimentale" que notre choix de ne pas remettre en cause notre distribution initiale des éléments est une bonne solution.

D'autre part, PABLAS doit allouer les coefficients de la matrice distribuée en termes de Degrés de Liberté, car jusqu'alors les structures étaient manipulées par les différentes étapes séquentielles d'EMILIO au niveau des nœuds du maillage. Cette expansion est basée sur le fait que les Degrés de Liberté d'un même nœud sont numérotés continuement (voir figure 5.1). Pour optimiser les accès mémoire, ces coefficients sont stockés sous la forme d'un vecteur dont la taille a été calculée au préalable par le module BLEND.

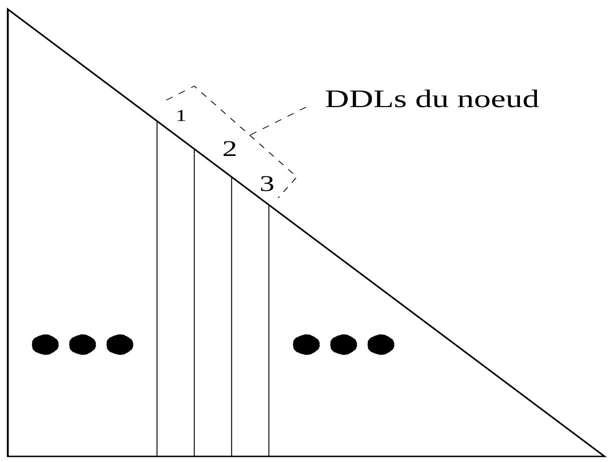

FIG. 5.1: Degrés de Liberté numérotés continuement pour un même nœud.

Après avoir alloué les coefficients de la matrice distribuée, chaque processeur prépare ses buffers d'envoi et son buffer de réception. Le nombre de buffers d'envoi est fixé par défaut car notre assemblage parallèle utilise un mode de communication asynchrone ; on utilise ces tampons (en nombre suffisant) pour ne pas rester "bloqué" par un envoi qui n'aurait pas été encore receptionné par son destinataire. Ce nombre de tampons a été fixé à 10 aprés différentes exécutions sur une collection de cas tests et après l'analyse des diagrammes de Gantt correspondants. En ce qui concerne les réceptions, un seul buffer suffit pour l'ensemble de l'assemblage et nous effectuons uniquement des réceptions bloquantes.

On va maintenant détailler le fonctionnement de notre assemblage parallèle et les choix effectués. On rappelle que l'assemblage parallèle doit gérer *deux activités concurrentes* : le calcul des matrices élémentaires des éléments attribués au processeur courant, ainsi que les communications contenant des contributions de mise à jour des blocs locaux de la matrice globale. Notre assemblage parallèle (module

PABLAS) doit traiter les deux types d'éléments qui peuvent être attribués au processeur courant par le module SACHA : les éléments non totalement locaux et les éléments totalement locaux.

Les notations pour l'algorithme de notre assemblage parallèle sont détaillées à la page suivante.

```
Pour le processeur p :
Pour e ∈ [1, Ne_nloc] Faire
    Si message(s) en attente Alors
        % étape de réception/mise à jour
        nmesg ← 0 ;
        Tant que nmesg < N et r_p > 0
        et message(s) encore en attente ;
            Recevoir message m ;
            A = A + CONTRIB_m ;
            r_p ← r_p − 1 ; nmesg ← nmesg + 1 ;
    % étape de calcul/émission
    Calculer matrice élémentaire M_e ;
    Préparer CONTRIB_e ;
    Envoyer CONTRIB_e ;
    A = A + CONTRIB_LOCALES_e ;
Pour e ∈ [1, Ne_loc] Faire
    Si message(s) en attente Alors
        nmesg ← 0 ;
        Tant que nmesg < N et r_p > 0
        et message(s) encore en attente ;
            Recevoir message m ;
            A = A + CONTRIB_m ;
            r_p ← r_p − 1 ; nmesg ← nmesg + 1 ;
    Calculer matrice élémentaire M_e ;
    A = A + CONTRIB_LOCALES_e ;
Tant que r_p > 0
    Recevoir message m ;
    A = A + CONTRIB_m ;
    r_p ← r_p − 1 ;
```

FIG. 5.2: Algorithme de l'assemblage parallèle pour une distribution 1D.

Notations :

- Ne_{nloc} : nombre d'éléments non totalement locaux pour processeur p,
- Ne_{loc} : nombre d'éléments totalement locaux pour processeur p,
- N : nombre maximum de messages à recevoir consécutivement,
- R : nombre de contributions à recevoir pour le processeur p,
- r_p : nombre de contributions extérieures déjà recues (initialisée à R).

Pour chaque processeur, *on commence par traiter les éléments non totalement locaux qui lui ont été attribués* (ligne 1). Avant de calculer la matrice élémentaire d'un élément non totalement local, on teste si un ou plusieurs messages ne sont pas en attente de réception. Si c'est le cas, on effectue une réception bloquante (on a fixé au préalable un nombre R maximum de messages à recevoir afin de réguler les activités de réception et de calcul du processeur), et on effectue les mises à jour dans les blocs locaux de la matrice avec les contenus des messages reçus (ligne 9). Sinon, on calcule la matrice élémentaire d'un élément non totalement local, on prépare les messages de contribution à envoyer aux processeurs concernés (ligne 13), puis on effectue les éventuelles additions locales de contributions, s'il y en a (ligne 15).

En ce qui concerne le calcul d'une matrice élémentaire, PABLAS fait appel au module ELAS qui lui renvoit une matrice élémentaire sous forme de vecteur, le problème étant que cette matrice est calculée avec la numérotation initiale du code OSSAU. Il est donc nécessaire de réordonner les coefficients de la matrice élémentaire pour tenir compte de la renumérotation calculée par le module ORDER. De plus, comme nous trions les numéros de nœuds d'un élément dans l'ordre croissant afin d'optimiser les accès à la structure de données (blocs de coefficients stockés sous forme de vecteur), il faut aussi tenir compte de ce tri pour le réordonnancement de la matrice élémentaire renvoyée par le module ELAS.

Pour les additions de contributions dans les blocs locaux de la matrice globale, on utilise les routines de calcul de type BLAS de niveau 1 (opérations entre vecteurs [48, 49]). D'autre part, pour réduire le nombre de communications, les messages sont bufferisés au niveau d'un élément. Cela signifie que pour un élément non totalement local, le processeur propriétaire de cet élément ne va pas envoyer un message par nœud non local, mais plutôt regrouper les nœuds non locaux appartenant à un même processeur dans un même message. Par exemple, pour la grille de la figure 4.1, en supposant que pour l'élément 2 les nœuds 17 et 15 soient situés sur le processeur i et les nœuds 25 et 24 situés sur le processeur j, si le processeur i est le propriétaire de l'élément 2, alors au lieu d'envoyer deux messages au processeur j concernant les nœuds 25 et 24, ces deux contributions seront bufferisées dans un même message à destination de j.

Lorsque le traitement des éléments non totalement locaux est terminé, *on traite alors les éléments totalement locaux*, ce qui signifie que l'on doit calculer les matrices élémentaires de ces éléments et ajouter leurs contributions uniquement dans les blocs locaux. Cette étape est donc complètement parallèle, et sans aucune communication entre processeurs. Néammoins, avant de calculer une telle matrice élémentaire, on teste encore si des messages contenant des contributions extérieures pour d'autres éléments ne sont pas en attente de réception (ligne 17), et si c'est le cas, ils sont traités en premiers (en effet, le processeur peut recevoir des contributions d'éléments appartenant à d'autres processeurs mais dont un ou plusieurs nœuds ont été attribués au processeur courant). De la même manière que pour les éléments non totalement locaux, on effectue une réception bloquante et on a déterminé au préalable un nombre R maximum de messages à recevoir afin de réguler les activités de réception et de calcul du processeur. *On a fait le choix de traiter en dernier les éléments totalement locaux afin de pouvoir recouvrir par les calculs des matrices élémentaires de ces éléments les communications induites par les éléments non totalement locaux*; on pourra constater dans le paragraphe 5.3 que ce choix est pertinent.

Enfin, lorsque le module PABLAS a fini de traiter tous les éléments attribués au processeur courant, il lance une réception bloquante et attend, si nécessaire, des messages afin de finir de mettre à jour les blocs locaux de la matrice K (ligne 27). Bien évidemment, cette réception ne sera lancée que si le compteur de messages encore à recevoir est non nul. Ce compteur de messages à recevoir a été intialisé par le module SACHA lors de l'attribution des éléments à l'ensemble des processeurs.

On utilise le même schéma d'exécution pour l'assemblage du second membre (module SMPABLAS), celui-ci travaillant avec des seconds membres élémentaires au lieu de matrices élémentaires. En ce qui concerne la distribution du vecteur second membre, nous avons décidé d'attribuer à chaque processeur les parties du second membre correspondants aux indices des blocs colonnes dont il est possesseur (voir module BLEND). Les distributions des blocs de la matrice et du second membre sont donc cohérentes l'une par rapport à l'autre, et les schémas de calculs et de communications seront les mêmes pour les deux assemblages.

Dans le paragraphe suivant, nous allons présenter des résultats pour un pseudo-assemblage de la matrice globale de rigidité avec un jeu de cas tests réguliers et irréguliers.

TAB. 5.1: Temps d'assemblage de pseudo matrices élémentaires.

Nom	Processeurs					Temps élémentaire
	4	8	16	32	64	
COLOGB30	3.50	1.93	0.98	0.56	0.32	0.01s
COLOGB75	8.79	4.64	2.38	1.36	0.80	-
CUBE39	139.52	69.71	35.15	17.89	9.02	-
BILLES	13.40	6.72	3.45	1.86	1.09	-
CEL3DM	37.32	18.72	9.55	4.92	2.72	-
COLOGB30	16.56	8.55	4.58	2.31	1.24	0.05s
COLOGB75	41.13	20.72	10.67	5.88	3.01	-
CUBE39	690.42	344.44	172.55	86.60	43.47	-
BILLES	65.30	32.72	16.45	8.40	4.46	-
CEL3DM	181.89	91.29	45.64	22.93	11.49	-
COLOGB30	34.42	16.77	8.43	4.28	2.23	0.1s
COLOGB75	80.90	40.60	20.67	10.68	5.48	-
CUBE39	1374.69	687.19	344.06	172.10	87.41	-
BILLES	130.34	65.22	32.91	16.56	8.32	-
CEL3DM	362.83	181.81	90.91	45.80	22.86	-

5.3 Validation expérimentale de l'assemblage

Ces expérimentations parallèles ont été réalisées sur le calculateur IBM SP2 du CINES (Montpellier, France) doté de 192 nœuds fins 120 MHz Power2SC (480 MFlops de puissance crête) disposant chacun de 256 MBytes de mémoire physique et d'un réseau d'interconnexion basé sur un switch TB3.

Elles ont été effectuées sur des jeux de test réguliers (CUBE39) et irréguliers provenant du CEA/CESTA (COLOGB30, COLOGB75, CEL3DM, BILLES). Nous considérons trois temps d'exécution de référence pour le calcul d'une matrice élémentaire : 0.01 seconde, 0.05 seconde et 0.1 seconde. Ces temps de calcul d'une pseudo-matrice élémentaire proviennent d'une analyse d'exécution de cas réels en séquentiel. Nous pouvons ainsi simuler dans cette partie le calcul d'une matrice élémentaire car le but est ici de valider expérimentalement notre algorithme d'assemblage avant de le coupler avec le module ELAS issu du code OSSAU original. Pour connaître les caractéristiques des cas tests irréguliers, on pourra se reporter aux chapitres 7 et 8. Les temps sont exprimés en secondes et sont des temps moyens sur les p processeurs pour un assemblage parallèle.

On peut constater dans le tableau de résultats 5.1 que sur les cas réguliers (CUBE39) et irréguliers (cas tests du CEA), les résultats obtenus par notre algorithme parallèle sont très "scalables", et ceci d'autant plus que le temps de calcul d'une matrice élémentaire est important : par exemple, pour le cas test COLOGB75 l'efficacité relative par rapport à 4 processeurs est pour 64 processeurs de 68,6 % pour un temps de calcul élémentaire de 0.01 s, de 85,4 % pour un même temps de 0.05 s, et enfin de 92,26 % pour un temps de 0.1 s. Cette observation est valable pour tous les exemples tests.

D'autre part, et si on regarde encore le cas test COLOGB75, lorsque le temps de calcul d'une pseudo matrice élémentaire est égal à 0.01 seconde, le temps d'un assemblage sur 16 processeurs (0.98 seconde) est 5 fois inférieur à un assemblage sur 16 processeurs avec un temps de calcul de matrice élémentaire égal à 0.05 seconde (4.58 secondes) et presque 10 fois inférieur pour un temps de calcul de 0.1 seconde (8.43 secondes).

Ces différents résultats permettent d'affirmer que la majeure partie du temps d'exécution est passée dans le calcul des matrices élémentaires et ce, quel que soit le maillage et le temps de calcul élémentaire. Si on prend par exemple le cas test COLOGB75 avec 4 processeurs et un temps de calcul de matrice élémentaire égal à 0.01 seconde, le temps séquentiel de calcul des matrices est alors égal à 31.5 secondes (3150 éléments $\times$ 0.01 seconde). Le temps que l'on obtient en parallèle sur 4 processeurs est égal à 8.79 secondes et si on divise le temps séquentiel par 4 nous obtenons 31.5 / 4 = 8 secondes environ ; donc, on peut constater que la majeure partie du temps est consacrée au temps de calcul des matrices élémentaires (environ 90 % du temps total sur un processeur pour ce cas test). Ceci confirme donc la qualité de notre distribution des éléments du maillage et le fait que la stratégie "guidant" notre assemblage parallèle permet de recouvrir complètement les communications correspondant aux éléments non totalement locaux par le calcul des matrices élémentaires des éléments totalement locaux.

Pour illustrer encore ce dernier fait, on donne ci-dessous des diagrammes d'activité de Gantt pour le COLOGB75 sur 16 processeurs. Dans ces diagrammes, la partie colorée en vert représente le traitement des éléments non totalement locaux, la partie colorée en jaune représente le traitement des éléments totalement locaux tandis que les autres parties colorées correspondent aux phases de communication et de traitement des messages.

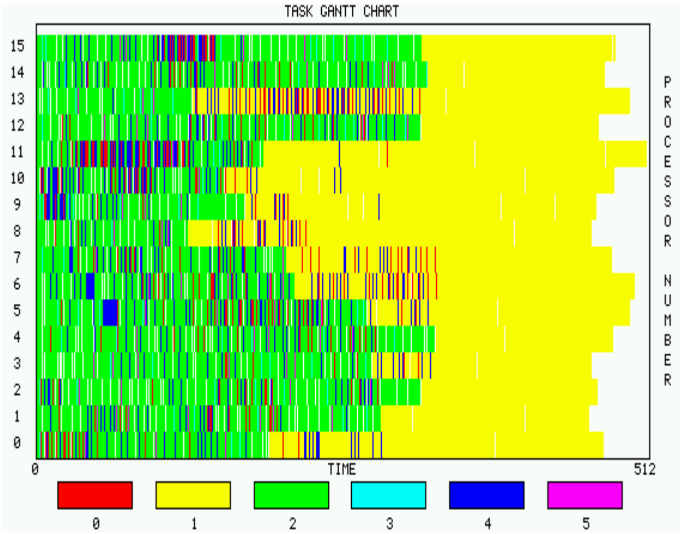

FIG. 5.3: Diagramme de Gantt pour COLOGB75 (0.01 s).

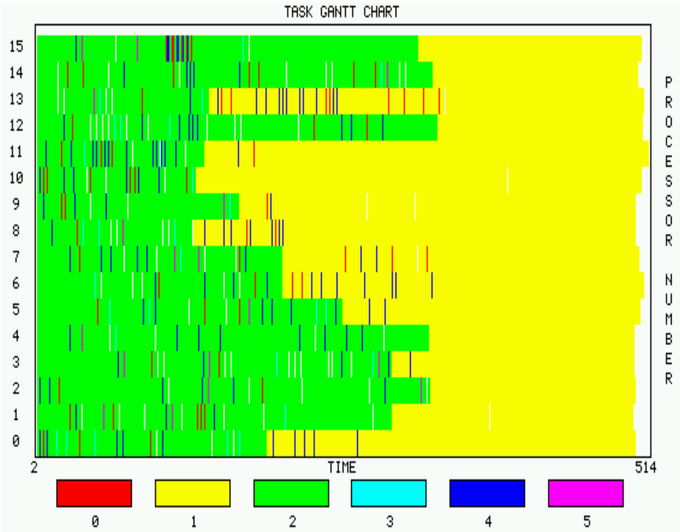

FIG. 5.4: Diagramme de Gantt pour COLOGB75 (0.05 s).

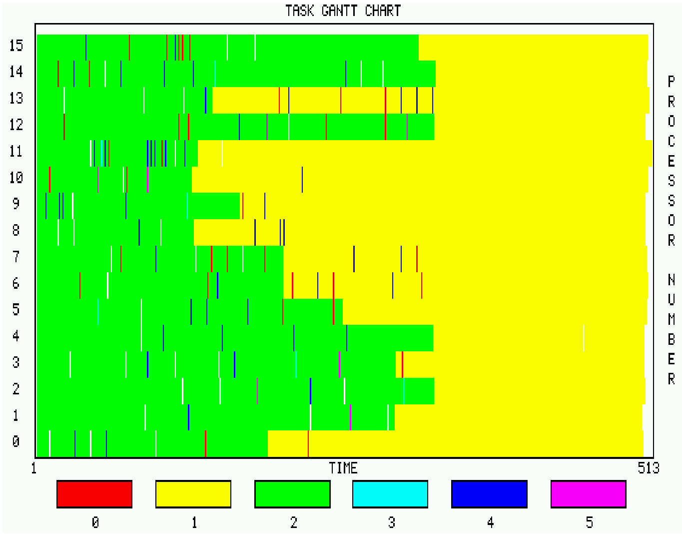

FIG. 5.5: Diagramme de Gantt pour COLOGB75 (0.1 s).

On peut voir ainsi sur ces diagrammes de Gantt 5.3, 5.4, 5.5 que les communications (traits dans la partie gauche des diagrammes) sont d'autant mieux recouvertes par les calculs locaux (partie jaune à droite des diagrammes) que le temps de calcul d'une pseudo matrice élémentaire est important. On voit également que le temps de traitement des éléments non totalement locaux diffère suivant les processeurs, mais que le temps de traitement global s'équilibre grâce au traitement des éléments totalement locaux.

Dans les chapitres suivants, nous allons maintenant présenter des résultats d'exécution de la chaîne complète EMILIO pour un ensemble de cas tests réels du CEA/CESTA, certains dépassant le million d'inconnues. Nous donnerons aussi une estimation mémoire nécessaire pour l'exécution du module ELAS et de la partie FORTRAN qui nous sert lors du lancement du calcul.

Troisième partie

Validation Expérimentale

Chapitre 6

Gestion de la mémoire

6.1 Gestion de la mémoire dans le projet EMILIO

La mémoire doit être utilisée pour tous les traitements de la bibliothèque réalisés en langage C (allocation dynamique), mais aussi pour les routines provenant du code OSSAU original et qui sont écrites en Fortran. En ce qui concerne les allocations faites par le code Fortran, on a supprimé l'allocation de la matrice globale et du second membre et gardé le reste des allocations en l'état. Néammoins, afin d'effectuer la gestion dynamique de la mémoire nécessaire aux étapes effectuées en C (Assemblage, Résolution, etc...), on n'attribue plus toute la mémoire à ce code Fortran (ce qui était le cas dans le code original) mais seulement la taille utile. Cette taille est déterminée par une pré-exécution du code OSSAU avec le cas test qui nous intéresse.

Chronologiquement, la partie du code OSSAU original est présente en mémoire pour tous les processeurs dès l'initialisation, puis le pré-traitement séquentiel de la chaîne EMILIO s'exécute sur l'ensemble des processeurs, les communications inter-modules s'effectuant donc au moyen d'allocations mémoire de structures (allocation dynamique en langage C). Ces différents modules vont allouer, dans la mesure du possible, en premier lieu les structures utilisées par les modules suivants (dans la chronologie de l'exécution) et ceci afin d'éviter au maximum la fragmentation mémoire. Le schéma 6.1 résume cette organisation mémoire.

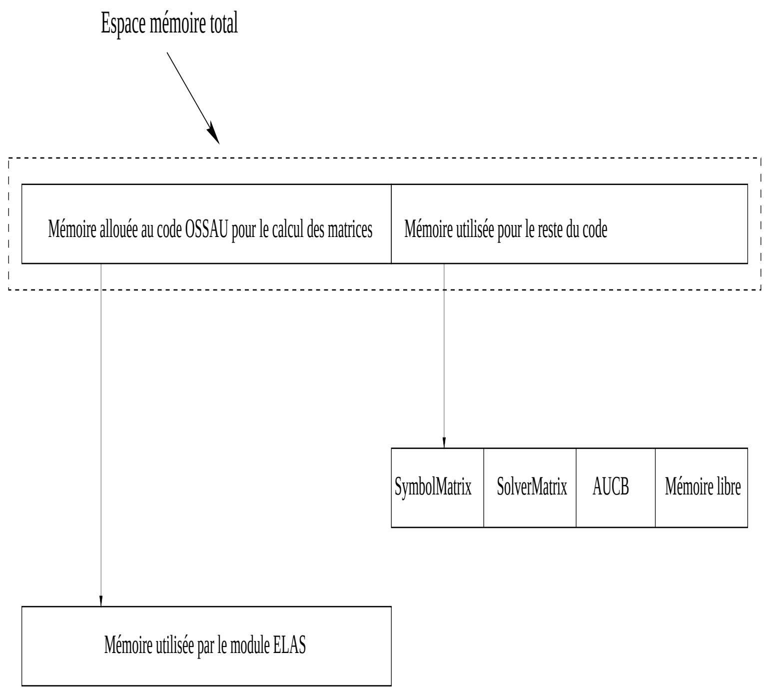

FIG. 6.1: L'utilisation de la mémoire

avec successivement :

- SymbolMatrix correspondant à l'espace mémoire pour stocker la matrice globale symbolique après la factorisation logique,
- SolverMatrix correspondant à l'espace mémoire pour stocker la matrice locale au processeur après l'assemblage,
- $AUCB$ correspondant à l'espace mémoire nécessaire pour stocker les aggrégations locales de contributions pour le solveur parallèle.

6.2 Tailles mémoire nécessaires pour le code OSSAU original

Dans le tableau suivant, on a résumé pour les différents cas tests que nous traiterons dans les chapitres suivants les tailles mémoires nécessaires pour le code OSSAU et donc pour le module ELAS de calcul des matrices élémentaires. Ces tailles sont des des tailles mémoire par processeur, car comme nous l'avons décrit dans les chapitres précédents, le code OSSAU original est séquentiel.

Tab. 6.1: Tailles mémoires en octets pour le code OSSAU.

Nom cas test	Taille mémoire (en octets) par processeur
COLOGB00	152448
COLOGB15	810512
COLOGB30	1558672
COLOGB75	3803072
CEL3DM	2918000
COUPOLE20T	1775520
COUPOLE450T	20038760
COUPOLE1500T	66604736
COUPOLE2000T	88778736
COUPOLE3000T	133127700

Chapitre 7

Le cas de Cologne

7.1 Description du Problème

7.1.1 Test de Cologne

Il s'agit d'une "languette" métallique formant une sorte de Z, disposée comme le montre la figure ci-dessous. Les nœuds de l'extrémité située dans le plan $Y = 0$ sont bloqués. Une force répartie non "suiveuse" parallèle à l'axe des Z est appliquée progressivement à l'autre extrémité. Le calcul est fait en grands déplacements et plasticité.

Les figures suivantes présentent des positions successives de la languette au cours du chargement appliqué progressivement : il s'agit des déformées vraies (non amplifiées). Ont été coloriées sur ces déformées les iso-valeurs du critère de Von Mises : les zones coloriées en rouge sont plastifiées.

La languette subit d'abord un effort de flexion principalement sur la partie proche de l'encastrement qui plastifie dans toute l'épaisseur jusqu'au premier coude (position 1 et 2). Si on était en plasticité parfaite, il y aurait à ce moment-là une "rotule plastique" totale et la matrice tangente serait singulière. Comme le matériau utilisé présente de l'écrouissage, le problème reste défini positif (mais proche de la singularité). Quand la languette dépasse une inclinaison de 45^o (position 3), les forces commencent à appliquer un effort de membrane qui vient s'opposer dans certains points de l'épaisseur à la contrainte de flexion. Une partie de cette épaisseur redevient élastique. Quand la languette est à 90^o (position 4) le chargement crée très majoritairement un effort de membrane. La languette est redevenue presque complètement élastique.

7.1.2 Tests de Cologne allongés en X

Ces tests consistent à allonger la languette précédente dans la direction de l'axe des X d'un certain facteur multiplicatif (15 pour le test COLOGB15, 75 pour le test COLOGB75). L'encastrement est le même et des forces réparties proportionnelles à l'allongement de la languette sont appliquées. Ainsi aux approximations d'effets de bord, les déplacements verticaux doivent être identiques à ceux de la languette initiale.

La figure suivante montre la languette allongée d'un facteur 15 dans 2 positions déformées successives au cours du chargement. Les isovaleurs du critère de Von-Mises sont coloriées (le rouge correspond aux zones ayant plastifiées). On remarque certains effets de bord.

7.2 Validation expérimentale du cas de Cologne

Cette validation expérimentale se décompose en 3 parties :

- la description des maillages des 4 tests du cas de Cologne fournis par le CEA,
- les temps pour la distribution des éléments,
- les temps moyens pour les étapes parallèles et pour une itération complète.

Tous ces tests ont été effectués sur l'IBM SP2 du CINES dont les caractéristiques ont été fournies au paragraphe 5.3. Tous les temps donnés sont en secondes.

7.2.1 Description des maillages

Pour chacun des 4 cas tests, nous fournissons le nombre d'éléments du maillage (Nb elts), le nombre de nœuds (Nb nœuds) et les caractéristiques de la matrice (nombre de colonnes (Nb colonnes), nombre de termes (NNZ_{L}) dans la matrice factorisée, nombre d'opérations pour factoriser OPC). Il y a ici 3 Degrés De Liberté par nœud.

TAB. 7.1: Description numérique des cas du test de Cologne.

Cas test	Nb elts	Nb nœuds	Nb colonnes	NNZ_{L}	OPC
COLOGB00	42	382	1146	94896	6.29626e+06
COLOGB15	630	3476	10428	4102335	1.65998e+09
COLOGB30	1260	6791	20373	9199719	4.5976e+09
COLOGB75	3150	16736	50208	26215551	1.51832e+10

7.2.2 Résultats

7.2.2.1 Distribution des éléments

Les temps séquentiels qui sont donnés dans le tableau suivant sont des temps moyens sur l'ensemble des processeurs (la distribution des éléments est une étape séquentielle dans l'enchaînement réel qui est exécutée par tous les processeurs (voir chapitre 6)).

Les valeurs absentes dans ce tableau correspondent à des cas où la taille du système est trop petite pour le nombre de processeurs considérés (impossibilité de distribuer des blocs de la matrice sur tous les processeurs).

Il est important de rappeler que l'on utilise pour l'instant une distribution 1D des blocs de la matrice, et que la renumérotation utilisée est basée sur la bibliothèque

SCOTCH [56] (dissections emboîtées et degré minimum approché avec contrainte de Halo [57]).

TAB. 7.2: Temps de distribution des éléments pour les cas du test de Cologne.

Cas test	Processeurs				
	4	8	16	32	64
COLOGB00	0.0008	0.0009	-	-	-
COLOGB15	0.0102	0.0122	0.0147	0.0162	-
COLOGB30	0.0149	0.0210	0.0249	0.0323	-
COLOGB75	0.0362	0.0391	0.0517	0.0718	0.1073

On constate sur le tableau de résultats 7.2 que les temps séquentiels de distribution des élément augmentent linéairement par rapport au nombre d'éléments du maillage, et logarithmiquement par rapport au nombre de processeurs utilisés. Notre algorithme de distribution est donc très peu coûteux en temps et même négligeable par rapport au temps consommé par les étapes parallèles exécutées itérativement (voir fin paragraphe 7.2.2.2).

7.2.2.2 Assemblage et résolution

Dans cette partie, nous analysons les étapes parallèles du logiciel EMILIO pour les tests du cas de Cologne. Ces exécutions se sont déroulées pour 75 incréments et plus de 400 itérations pour chaque cas test. Les temps donnés sont des temps moyens sur l'ensemble des itérations et sur l'ensemble des processeurs.

Pour les durées d'assemblage et de résolution, on a choisi de distinguer deux phases : les incréments précédents la phase de plastification et les incréments suivants la phase de plastification.

En ce qui concerne le nombre de processeurs utilisés pour les tests, nous avons exécuté les 75 incréments complets jusqu'à 32 processeurs pour les deux cas intermédiaires, et jusqu'à 64 processeurs pour le cas COLOGB75. Pour le test COLOGB00, vue la petite taille du système, nous n'avons exécuté les 75 incréments que sur 4 et 8 processeurs.

On rappelle que les différentes étapes constituant une itération de résolution sont :

– l'assemblage du second membre,

– l'assemblage de la matrice de rigidité K,

– la mise aux conditions limites,

– la factorisation numérique,

– la résolution du système,

– le calcul du résidu.

Dans les tableaux qui suivent, le temps d'assemblage est celui de la matrice et la rubrique "Autres" représente les temps cumulés de l'assemblage du second membre, de la mise aux conditions limites et du calcul du résidu. Il est important de rappeler que les temps donnés dans les tableaux suivants sont des temps moyens sur un certain nombre d'itérations et peuvent donc être différents de ceux donnés dans les Annexes (notamment pour la factorisation et la résolution) car ces derniers sont des temps maximum pour une seule résolution du système.

TAB. 7.3: Temps des étapes parallèles pour une itération de COLOGB00.

Etape	Processeurs			
	4		8	
Assemblage	0.313	0.310	0.176	0.168
Factorisation	0.041		0.034	
Résolution	0.019		0.013	
Autres	0.06		0.07	
Durée Itération	0.433	0.430	0.293	0.285

Pour ce premier cas test COLOGB00, le faible nombre d'éléments et la petite taille du système ne peuvent pas donner d'informations très fiables sur la qualité de l'assemblage parallèle, mais on remarque déjà que les étapes parallèles d'assemblage et de factorisation constituent la partie la plus coûteuse en temps d'une itération. De plus, on peut constater le faible écart en terme de temps entre l'assemblage correspondant aux incréments de pré-plastification (valeur de gauche) et l'assemblage des incréments post-plastification (valeur de droite).

Tab. 7.4: Temps des étapes parallèles pour une itération de COLOGB15.

Etape	Processeurs							
	4		8		16		32	
Assemblage	4.293	4.219	2.201	2.202	1.225	1.222	0.656	0.654
Factorisation	3.13		1.61		1.11		0.96	
Résolution	0.082		0.059		0.058		0.057	
Autres	0.09		0.08		0.07		0.07	
Durée Itération	7.595	7.521	3.950	3.951	2.463	2.460	1.743	1.741

Pour ce cas test COLOGB15, on constate que les temps d'assemblage de la matrice globale de rigidité sont plus importants que ceux de la factorisation jusqu'à 16 processeurs. Cela confirme bien que l'assemblage est une étape très coûteuse pour ce type d'application. En ce qui concerne l'écart en temps entre la phase de post-plastification et la phase de pré-plastification, elle est tout aussi "négligeable" que pour le cas test COLOGB00.

Tab. 7.5: Temps des étapes parallèles pour une itération de COLOGB30.

Etape	Processeurs							
	4		8		16		32	
Assemblage	8.052	8.029	4.437	4.427	2.287	2.279	1.227	1.222
Factorisation	5.43		3.10		2.01		1.75	
Résolution	0.161		0.112		0.080		0.104	
Autres	0.11		0.10		0.09		0.09	
Durée Itération	13.753	13.730	7.749	7.739	4.467	4.459	3.171	3.166

Pour ce cas test qui comporte plus de 20000 Degrés de Liberté, on a les mêmes comportements que ceux décrits pour le COLOGB15.

TAB. 7.6: Temps des étapes parallèles pour une itération de COLOGB75.

Etape	Processeurs									
	4		8		16		32		64	
Assemblage	20.560	20.560	10.503	10.470	5.542	5.544	2.926	2.934	1.576	1.587
Factorisation	17.576		9.236		5.459		3.349		2.606	
Résolution	0.359		0.250		0.179		0.168		0.099	
Autres	0.15		0.14		0.14		0.13		0.11	
Durée Itération	38.645	38.645	20.129	20.096	11.320	11.322	6.573	6.581	4.391	4.402

Pour ce cas COLOGB75, qui est le plus grand cas test du challenge de Cologne que nous avions à notre disposition et qui comporte environ 50000 Degrés de Liberté, on constate encore que l'assemblage est plus coûteux que la factorisation jusqu'à 16 processeurs. Par la suite, les choses s'inversent naturellement. De plus, on constate que l'on obtient bien, conformément aux exemples précédents, *une très bonne scalabilité de l'assemblage et une bonne scalabilité globale* pour un nombre de processeurs inférieur ou égal à 64, et ceci, malgré une distribution 1D des blocs de la matrice.

Pour ce qui est de l'assemblage parallèle, ces tests corroborent les observations du chapitre 5 et le fait que le recouvrement calcul/communication est très efficace dans l'implémentation proposée.

On constate toujours que la majeure partie du temps pris par une itération est pris par l'assemblage et la factorisation et que les autres étapes sont négligeables. De même, l'écart de temps assez peu important constaté dans les cas tests COLOGB00, COLOGB15, COLOGB30 entre les itérations des incréments pré-plastification et les itérations des incréments post-plastification, se retrouve dans les résultats obtenus pour ce cas test. Cette dernière remarque nous renforce dans notre décision de ne pas remettre en cause notre distribution initiale des éléments.

Enfin, pour terminer cette analyse, il est intéressant de comparer le coût du prétraitement séquentiel pour 64 processeurs (18.43 secondes réparties en 17.69 secondes pour la renumérotation ; 0.24 seconde pour la factorisation symbolique ; 0.40 seconde pour la distribution de la matrice ; 0.107 seconde pour la distribution des éléments) par rapport au coût moyen d'une itération (4.4 secondes au pire). On constate donc que ce prétraitement sera rapidement amorti dès que le nombre d'itérations est raisonnablement grand. Cependant, l'utilisation à terme d'un partitionneur de graphes parallèle semble utile pour diminuer le coût de la phase de renumérotation.

Pour finir cette étude, on peut faire la remarque suivante : si dans les cas tests présentés ci-dessus, l'écart de temps C.P.U. entre les phases linéaires et non-linéaires n'est pas significatif, cela peut sans doute s'expliquer par le fait que l'on a ici 2 non linéarités couplées : plasticité et grand déplacement. Le coût supplémentaire des calculs après plastification est "négligeable" par rapport au surcoût dû aux grands déplacements (différence entre les matrices K_L et K_T du paragraphe 1.1.2.6.

Chapitre 8

Composite Tissé 3D

8.1 Description du Problème

Dans l'essai suivant, on cherche à identifier (par comparaison calcul-essai) le comportement élasto-plastique de la matrice (matériau jouant le rôle de liant) d'un matériau composite tissé 3D (qui pourrait être représentatif du comportement du matériau après un délaminage complet de la fibre et de la matrice). Le morceau de matrice (formé de 5 cellules périodiques de base) est soumis à un essai de flexion. La face arrière est encastrée et des forces ponctuelles sont appliquées sur l'avant de la pièce selon la figure suivante. Dans cet essai une partie de la matrice plastifie.

Les 3 figures suivantes présentent respectivement le maillage de la pièce sous 2 vues différentes, et le résultat du calcul à savoir les isovaleurs du critère de Von-Mises sur géométrie déformée.

8.2 Validation expérimentale des cas CEL3DM et CEL3DMB20

8.2.1 Description des maillages

Nous donnons dans ce paragraphe les caractéristiques des deux cas tests : nombre d'éléments, nombre de nœuds et les caractéristiques des matrices associées. Le nombre de Degrés de Liberté par nœud est égal à 3.

TAB. 8.1: Description du maillage du cas test CEL3DM.

Cas test	Nb elts	Nb nœuds	Nb colonnes	NNZ_{L}	OPC
CEL3DM	14400	10360	31080	8743689	3.33345e+09

Il est intéressant de remarquer que dans ce cas test, le nombre d'éléments est supérieur au nombre de nœuds du maillage.

TAB. 8.2: Description du maillage du cas test CEL3DMB20.

Cas test	Nb elts	Nb nœuds	Nb colonnes	NNZ_{L}	OPC
CEL3DMB20	14400	45832	137496	96269787	1.09584e+11

La différence entre les deux maillages réside dans le fait que CEL3DM est un maillage constitué de briques à 8 nœuds et de pentaèdres à 6 nœuds (interpolation linéaire), alors que CEL3DMB20 est constitué de briques à 20 nœuds et de pentaèdres à 15 nœuds (interpolation quadratique). Ils ont cependant le même nombre d'éléments.

8.2.2 Résultats

8.2.2.1 Distribution des éléments

Les temps de distribution que nous présentons dans le tableau 8.3 sont exprimés en secondes et sont des temps moyens sur l'ensemble des processeurs (SP2 du CINES).

TAB. 8.3: Temps de distribution des éléments pour le cas test CEL3DM.

Cas test	Processeurs				
	4	8	16	32	64
CEL3DM	0.1015	0.1314	0.2142	0.2428	0.4023

TAB. 8.4: Temps de distribution des éléments pour le cas test CEL3DMB20.

Cas test	Processeurs	
	32	64
CEL3DMB20	0.342	0.761

Les temps que l'on obtient sur ces distributions des éléments confirment ce que l'on avait pu constater dans le chapitre 7, à savoir que cette étape séquentielle est très peu coûteuse en temps.

8.2.2.2 Assemblage et résolution

Pour ces cas tests, nous avons réalisé 2 incréments et 5 itérations. Les temps donnés sont toujours des temps moyens sur l'ensemble des processeurs. D'autre part, nous ne distinguerons pas les deux phases de pré-plastification et de post-plastification, car sur un aussi petit nombre d'incréments il est impossible de discriminer ces 2 phases.

TAB. 8.5: Temps des étapes parallèles pour une itération de CEL3DM.

Nom étape	Processeurs				
	4	8	16	32	64
Assemblage	65.853	32.080	15.870	8.873	4.613
Factorisation	6.74	3.72	1.88	0.92	0.74
Résolution	0.20	0.11	0.07	0.12	0.10
Autres	1.2	0.9	0.9	0.8	0.8
Durée Itération	73.993	36.810	18.720	10.713	6.463

TAB. 8.6: Temps des étapes parallèles pour une itération de CEL3DMB20.

Nom étape	Processeurs	
	32	64
Assemblage	11.049	5.468
Factorisation	37.787	22.749
Résolution	3.952	0.686
Autres	3.5	2.8
Durée Itération	56.288	31.703

On constate pour le cas CEL3DM que l'assemblage "coûte" beaucoup plus cher que la factorisation, de l'ordre de 6 à 7 fois plus cher suivant le nombre de processeurs. Cela s'explique par le fait que le problème est de taille relativement petite (31000 colonnes) avec un nombre d'éléments finis assez grand (14400). La scalabilité de l'assemblage est alors meilleure que celle de la factorisation/résolution. Malgré tout, on observe une bonne scalabilité globale pour l'ensemble des tests.

Pour ce qui est du cas CEL3DMB20, on a des résultats d'ensemble identiques au cas précédent avec cependant un phénomène de "swap" pour 32 processeurs en particulier lors de la résolution. D'autre part, on voit bien que l'efficacité de l'assemblage est encore meilleure pour le CEL3DMB20 car le coût de calcul élémentaire d'un élément dans ce maillage est nettement plus important que dans le CEL3DM initial.

Enfin, dans ces cas tests, et contrairement à l'ensemble de ceux présentés précédemment, il y a deux types d'éléments finis différents (des briques et des pentaèdres), et la différence de temps de calcul des matrices élémentaires n'a pas été prise en compte dans la phase de distribution. Cette extension de critère (temps de calcul des matrices élémentaires plutôt que nombre de matrices élémentaires) sera à réaliser pour améliorer encore les performances pour ce type de maillages hétérogènes.

Chapitre 9

Coupole sous chargement "linéique"

9.1 Description du Problème

Il s'agit d'une coupole à plusieurs pentes qui subit un chargement "linéique". Seule une demi-géométrie est étudiée : le plan de symétrie choisi étant le plan $X0Z$. Le chargement est parallèle à l'axe $0Z$ sur les nœuds du plan de coupe composant la 2^{ieme} pente. Tant que l'on reste élastique linéaire, cette étude peut être réalisée en 2D Série de Fourier. Dès que l'on plastifie, ce n'est plus le cas et il faut passer au 3D. La page suivante montre la géométrie et les isovaleurs du critère de Von-Mises sur la géométrie déformée (déformée amplifiée).

Dans la phase linéaire, une comparaison a pu être faite avec des calculs 2D série de Fourier. Les maillages 3D étudiés font été obtenus par extrusion d'une "coupe" 2D avec un nombre de "couches" en Θ variables. Le nombre de "couches" de cette extrusion servira à nommer les différents cas tests. Par exemple, un cas avec 450 couches sera nommé COUPOLE450T par la suite.

9.2 Validation expérimentale du cas Coupole

9.2.1 Description des maillages

Nous détaillons dans le tableau 9.1 les caractéristiques des 5 cas tests de type Coupole que nous avons construits par la procédure présentée dans le paragraphe précédent. Le nombre de Degrés de Liberté par nœud est ici aussi égal à 3, et les éléments sont des briques à 20 nœuds.

TAB. 9.1: Description des maillages des cas tests Coupole.

Cas test	Nb elts	Nb nœuds	Nb colonnes	NNZ_{L}	OPC
COUPOLE20T	840	4581	13743	5988186	2.75959e+09
COUPOLE450T	18900	99611	298833	168699087	1.09519e+11
COUPOLE1500T	63000	331661	994983	578122029	3.79847e+11
COUPOLE2000T	84000	442161	1326483	758621313	5.00924e+11

9.2.2 Résultats

9.2.2.1 Distribution des éléments

Dans ce paragraphe, nous donnons les temps de distribution des éléments pour les différents cas tests de type Coupole. Pour les gros cas tests (un million de Degrés de Liberté et plus), les exécutions ont été réalisées exclusivement avec 64 processeurs afin d'éviter les phénomènes de swapping.

TAB. 9.2: Temps de distribution des éléments pour les cas tests Coupole.

Nom cas	Processeurs				
	4	8	16	32	64
COUPOLE20T	0.0115	0.0133	0.0169	0.0211	-
COUPOLE450T	-	-	0.2678	0.3452	0.5343
COUPOLE1500T	-	-	-	-	1.4539
COUPOLE2000T	-	-	-	-	1.8962

Dans le tableau 9.2, on constate encore que même pour les très gros cas tests qui avoisinent les deux millions de Degrés de Liberté, la distribution des éléments est très peu coûteuse en temps et reste linéaire par rapport au nombre d'éléments du maillage.

9.2.2.2 Assemblage et résolution

TAB. 9.3: Temps des étapes parallèles pour une itération de COUPOLE20T.

Nom étape	Processeurs			
	4	8	16	32
Assemblage	5.236	2.742	1.707	0.846
Factorisation	3.291	2.566	1.781	1.328
Résolution	0.105	0.054	0.047	0.077
Autres	0.29	0.186	0.173	0.161
Durée Itération	8.922	5.548	3.708	2.412

TAB. 9.4: Temps des étapes parallèles pour une itération de COUPOLE450T.

Nom étape	Processeurs		
	16	32	64
Assemblage	35.160	14.666	7.603
Factorisation	44.73	21.527	11.164
Résolution	0.534	0.462	0.334
Autres	8.33	7.67	7.18
Durée Itération	88.734	44.325	26.281

TAB. 9.5: Temps des étapes parallèles pour une itération de COUPOLE1500T.

Nom étape	Processeurs
	64
Assemblage	40.769
Factorisation	60.302
Résolution	1.363
Autres	40.302
Durée Itération	142.736

TAB. 9.6: Temps des étapes parallèles pour une itération de COUPOLE2000T.

Nom étape	Processeurs
	64
Assemblage	99.383
Factorisation	112.724
Résolution	1.978
Autres	60.388
Durée Itération	274.373

On donne dans les tableaux 9.3, 9.4, 9.5 et 9.6 les temps moyens pour les étapes parallèles pour tous les cas tests. Ces temps ont été obtenus sur 2 incréments avec un nombre d'itérations variant entre 2 et 11.

Comme pour les études de cas précédentes faites aux chapitres 7 et 8, on constate, malgré les légers phénomènes de "swap" pour les cas les plus gros, que :

- la factorisation parallèle redevient l'étape la plus coûteuse à partir de 16 processeurs,
- l'assemblage parallèle a une très bonne scalabilité,
- et que la scalabilité globale de l'enchaînement est tout à fait satisfaisante.

Néammoins, il est intéressant de remarquer que, dans ces cas tests dont la taille dépasse les 300.000 Degrés de Liberté, la partie Autres qui contient la mise à jour du déplacement total, l'assemblage parallèle du second membre, la mise aux conditions limites et le calcul du résidu semble prendre presque autant de temps que les phases parallèles d'assemblage de la matrice de rigidité et de résolution du système. Cependant, comme ces phases s'exécutent après les phases parallèles, des phénomènes de "swap" peuvent apparaître car l'allocation des coefficients a été faite précédemment lors de la 1ère itération.

On traite ainsi sur 64 processeurs type Power2SC 1 itération d'un problème à près de 300.000 Degrés de Liberté en une trentaine de secondes, 1 itération d'un problème à près de 1 million de Degrés de Liberté en moins de 3 minutes, et 1 itération d'un problème à près de 1.5 million de Degrés de Liberté en 4 minutes 30.

A titre de comparaison, le solveur initial séquentiel (avec pagination sur disque) conduisait pour le cas COUPOLE450T à un temps C.P.U. de 1600 secondes par itération. Ce solveur ne pouvait dépasser les 720 couches (soit 480000 Degrés de Liberté) à cause de la saturation de l'espace disque de pagination, une itération coûtant alors 2450 secondes.

Conclusion générale et perspectives

La contribution de cette thèse concerne la mise en œuvre hautes performances d'applications conduisant à la résolution de très grands systèmes linéaires creux ; on vise ainsi des problèmes discrétisés par éléments finis 3D pour lesquels le nombre de Degrés de Liberté dépasse largement le million. D'autre part, on a considéré des types d'applications, comme la mécanique des structures avec phénomène de plastification et/ou grands déplacements, où la discrétisation à utiliser conduit à des difficultés de comportement numérique qui rendent nécessaire l'utilisation de solveurs creux directs, ceux-ci étant reconnus par ailleurs pour leur robustesse numérique. La taille des données manipulées et les temps de traitement séquentiels relatifs à ce cadre applicatif rendent incontournable l'utilisation du parallélisme à une grande échelle. Notre contribution a porté alors sur deux domaines.

La première a consisté à participer activement à la conception et à la réalisation d'une bibliothèque algorithmique complète hautes performances de résolution par méthodes directes pour des systèmes linéaires creux de grande taille de type éléments finis. Ce projet (EMILIO) s'est fait au sein de l'équipe ALiENor du LaBRI dans le cadre d'un contrat de Recherche et Développement avec le CEA/CESTA. Le travail réalisé a consisté à définir une stratégie de distribution des éléments du maillage induite par celle des blocs de la matrice de rigidité, et à proposer un schéma de traitement performant des éléments ainsi distribués pour un algorithme d'assemblage parallèle. L'ensemble a été validé algorithmiquement et sur des tests industriels en vraie grandeur du CEA/CESTA. On obtient en particulier un assemblage parallèle qui a une très bonne scalabilité pour des maillages réguliers ou irréguliers.

La deuxième contribution a consisté en l'intégration dans un code industriel existant du CEA/CESTA (code OSSAU de mécanique des structures) de l'ensemble des outils algorithmiques séquentiels et parallèles développés par l'équipe ALiENor dans le cadre du projet EMILIO. Cette intégration, qui avait nécessité préalablement un travail de “profiling” du code séquentiel existant, a été validée et a débouché sur des résultats de bonne qualité pour divers problèmes de mécanique des structures dont certains dépassent en taille le million de Degrés de Liberté pour des problèmes 3D. Les résultats ainsi obtenus sur l'ensemble des étapes parallèles de l'enchaînement global montrent une bonne scalabilité, et permettent de dire que l'objectif industriel recherché est effectivement atteint.

L'évolution de ce travail va consister tout d'abord en l'intégration de nouvelles possibilités algorithmiques concernant la bibliothèque EMILIO.

Il est prévu tout d'abord de remplacer le solveur actuel, qui est basé sur une distribution bloc 1D, par le solveur actuellement en fin de réalisation au sein de l'équipe ALiENor et qui est basé sur une distribution mixte 1D/2D. Cela permettra l'utilisation d'un plus grand nombre de processeurs car la scalabilité de la factorisation/résolution s'en trouvera améliorée. Dans ce cadre, nous aurons à adapter la distribution des éléments et l'assemblage parallèle pour tenir compte de cette nouvelle distribution des blocs de la matrice de rigidité à assembler. On améliorera aussi le critère d'équilibrage pour la distribution des éléments dans le cas de maillages hétérogènes (coût de calcul des matrices élémentaires plutôt que leur nombre).

Une autre évolution essentielle consistera à valider l'ensemble de la chaîne algorithmique sur des architectures de machine basée sur des réseaux de nœuds SMP. Là encore, il faudra adapter la distribution pour le solveur (travaux en cours), et la stratégie pour l'assemblage. Ceci permettra alors l'utilisation du nouveau calculateur dont le CEA/DAM a fait l'acquisition.

Enfin, on pourra utiliser l'ensemble de ces outils dans le cadre d'une autre application d'électromagnétisme, où l'on est amené à traiter de très grands systèmes, obtenus par un couplage éléments finis – équations intégrales, constitués d'une grande partie creuse et d'une partie dense (système en variables complexes symétriques mais non hermitien d'où l'intérêt pour les solveurs directs).

Annexes

Annexe A

Conditions expérimentales pour les cas tests relatifs au chapitre 3

Dans cette annexe, nous présentons une collection de matrices creuses provenant de la collection Rutherford-Boeing [21], du projet PARASOL ESPRIT IV LTR No. 20160 (`http ://www.parallab.uib.no/projects/parasol/`) et du CEA/CESTA (Cologne 3D). Ces matrices ont servi aux expérimentations parallèles pour les étapes algorithmiques décrites au chapitre 3.

Le développement et la validation ont été effectués sur le calculateur IBM SP2 du LaBRI disposant de 16 nœuds. Les expérimentations parallèles en vraie grandeur ont été réalisées sur le calculateur IBM SP2 du CINES doté de 192 nœuds fins 120 MHz Power2SC (480 MFlops de puissance crête) disposant chacun de 256 MBytes de mémoire physique et d'un réseau d'interconnexion basé sur un switch TB3. Les expérimentations séquentielles des phases de pré-traitement ont également été effectuées sur un nœud de ce calculateur IBM SP2.

Les caractéristiques des matrices sont données dans la table A.1 ; la renumérotation est issue du logiciel SCOTCH (dissections emboîtées + degré minimum approché exploitant la contrainte de Halo [57]). Ces caractéristiques ont été obtenues à l'aide d'une factorisation symbolique *scalaire* pour une renumérotation sans amalgamation dans la partie degré minimum : NNZ_{A} est le nombre de termes extra-diagonaux dans la partie triangulaire inférieure de la matrice A, NNZ_{L} est le nombre de termes extra-diagonaux dans la matrice factorisée L et OPC est le nombre d'opérations pour factoriser.

Nom	Taille	NNZ_A	NNZ_L	OPC	Description
GRID511	261121	1041420	1.202166e+07	2.565341e+09	Maillage 2D régulier
GRID767	588289	2348556	2.979676e+07	8.745496e+09	Maillage 2D régulier
GRID1023	1046529	4179980	5.615708e+07	2.083481e+10	Maillage 2D régulier
CUBE31	29791	361890	8.346406e+06	5.525167e+09	Maillage 3D régulier
CUBE39	59319	730778	2.210534e+07	2.240674e+10	Maillage 3D régulier
CUBE47	103823	1290898	4.828456e+07	6.963850e+10	Maillage 3D régulier
144	144649	1074393	4.696077e+07	5.659395e+10	
598A	110971	741934	2.590574e+07	1.876964e+10	
BCSSTK30	28924	1007284	4.309003e+06	1.188915e+09	Rutherford-Boeing
BCSSTK32	44609	985046	5.239146e+06	1.162900e+09	Rutherford-Boeing
M14B	214765	1679018	6.236747e+07	6.112540e+10	
OCEAN	143437	409593	2.029997e+07	1.301477e+10	
BBMAT	38744	1274141	1.716094e+07	1.250040e+10	Rutherford-Boeing
TOOTH	78136	452591	1.031143e+07	6.267094e+09	Rutherford-Boeing
SMDOOR	162610	3873534	2.541937e+07	1.530774e+10	PARASOL
OILPAN	73752	1761718	8.912337e+06	2.984944e+09	PARASOL
BMWCRA_1	148770	5247616	6.597301e+07	5.701988e+10	PARASOL
INVEXTR1	30412	906915	7.256566e+06	3.766788e+09	PARASOL
MIXTANK	29957	982542	9.280247e+06	7.316933e+09	PARASOL
MT1	97578	4827996	3.114873e+07	2.109265e+10	PARASOL
QUER	59122	1403689	9.118592e+06	3.280680e+09	PARASOL
SHIP001	34920	2304655	1.427916e+07	9.033767e+09	PARASOL
SHIP003	121728	3982153	5.872912e+07	8.008089e+10	PARASOL
SHIPSEC5	179860	4966618	5.649801e+07	6.952086e+10	PARASOL
SHIPSEC8	114919	3269240	3.572761e+07	3.684269e+10	PARASOL
THREAD	29736	2220156	2.404333e+07	3.884020e+10	PARASOL
X104	108384	5029620	2.634047e+07	1.712902e+10	PARASOL
BMW3_2	227362	5530634	4.420244e+07	3.007981e+10	PARASOL
CRANKSG1	52804	5280703	3.142730e+07	3.007141e+10	PARASOL
CRANKSG2	63838	7042510	4.190437e+07	4.602878e+10	PARASOL
COLOGB15	10428	701292	3.371109e+06	1.495460e+09	Cologne 3D
COLOGB30	20373	1394292	7.849464e+06	4.359803e+09	Cologne 3D
COLOGB75	50208	3473292	2.248613e+07	1.532035e+10	Cologne 3D

TAB. A.1: Description des matrices tests.

Annexe B

Temps d'exécution des modules séquentiels de factorisation symbolique et de distribution de la matrice

La table B.1 regroupe les temps d'exécution séquentiels en secondes de l'étape de repartitionnement et de distribution pour un nombre de processeurs variant entre 2 et 32, ainsi que le temps de la factorisation symbolique par bloc. On notera les très faibles coûts en temps de ces étapes.

Nom	Processeurs					Temps_FS
	2	4	8	16	32	
GRID511	0.35	0.37	0.43	0.52	0.66	0.83
GRID767	0.73	0.76	0.87	1.06	1.32	1.77
GRID1023	1.43	1.50	1.75	2.20	2.64	3.31
CUBE31	0.09	0.13	0.17	0.27	0.38	0.27
CUBE39	0.17	0.26	0.36	0.71	0.95	0.58
CUBE47	0.51	0.71	0.92	1.80	2.46	1.35
144	2.10	2.92	3.71	4.55	5.99	1.63
598A	1.50	1.86	2.86	3.19	4.17	1.27
BCSSTK30	0.05	0.08	0.09	0.11	0.13	0.47
BCSSTK32	0.14	0.18	0.20	0.24	0.30	0.55
M14B	2.27	3.10	4.62	5.80	6.80	2.43
OCEAN	2.38	2.97	3.62	4.44	5.65	1.29
BBMAT	0.14	0.29	0.41	0.62	0.77	0.69
TOOTH	1.47	1.73	2.19	2.72	2.95	0.96
SMDOOR	0.39	0.44	0.51	0.59	0.73	1.93
OILPAN	0.20	0.21	0.22	0.24	0.30	0.88
BMWCRA_1	0.19	0.27	0.52	0.82	0.97	2.59
INVEXTR1	0.41	0.62	0.76	0.92	1.35	0.62
MIXTANK	0.24	0.62	0.76	0.92	1.05	0.61
MT1	0.10	0.16	0.22	0.27	0.31	1.80
QUER	0.13	0.15	0.16	0.17	0.20	0.69
SHIP001	0.05	0.06	0.09	0.14	0.17	0.88
SHIP003	0.41	0.67	1.04	1.50	1.86	2.05
SHIPSEC5	0.74	0.84	1.19	1.46	1.87	2.43
SHIPSEC8	0.41	0.64	0.87	1.16	1.31	1.62
THREAD	0.06	0.34	0.37	0.42	0.46	0.87
X104	0.17	0.22	0.26	0.30	0.35	1.95
BMW3_2	0.54	0.84	0.91	1.08	1.24	3.12
CRANKSG1	0.05	0.09	0.22	0.34	0.44	1.99
CRANKSG2	0.05	0.12	0.25	0.43	0.50	2.55
COLOGB15	0.02	0.03	0.04	0.05	0.06	0.05
COLOGB30	0.02	0.05	0.07	0.10	0.13	0.10
COLOGB75	0.07	0.11	0.15	0.21	0.25	0.24

TAB. B.1: Temps d'exécution des étapes de partitionnement/distribution et factorisation symbolique pour une distribution 1D.

Annexe C

Temps de factorisation parallèle pour une distribution bloc 1D

Les tables C.1 et C.2 présentent les performances de la factorisation parallèle par blocs pour notre distribution 1D (le **temps** est donné en secondes et la puissance en gigaflops) pour les cas réguliers (grilles 2D et cubes 3D) et pour les cas irréguliers. Les tirets correspondent à des exécutions faussées par des phénomènes de "swapping" ou d'allocation mémoire impossible.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
GRID511	**8.06** (0.32)	**4.17** (0.62)	**2.36** (1.09)	**1.47** (1.75)	**1.16** (2.21)	**0.97** (2.64)
GRID767	-	**11.94** (0.73)	**6.49** (1.35)	**3.95** (2.21)	**2.79** (3.14)	**2.45** (3.56)
GRID1023	-	-	**18.23** (1.14)	**8.22** (2.53)	**5.36** (3.88)	**5.42** (3.84)
CUBE31	**11.68** (0.47)	**6.38** (0.87)	**3.77** (1.47)	**2.72** (2.03)	**2.42** (2.28)	**2.43** (2.28)
CUBE39	**42.54** (0.53)	**23.24** (0.96)	**13.17** (1.70)	**8.13** (2.76)	**6.83** (3.28)	**5.32** (4.21)
CUBE47	-	**78.02** (0.89)	**37.54** (1.85)	**22.90** (3.04)	**16.60** (4.20)	**14.78** (4.71)

TAB. C.1: Temps d'exécution (vitesse en Gflops) de la factorisation parallèle pour une distribution 1D pour des cas réguliers.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
144	-	**70.64** (0.80)	**38.73** (1.46)	**22.45** (2.52)	**13.48** (4.20)	**11.42** (4.95)
598A	**54.90** (0.34)	**26.55** (0.71)	**14.42** (1.30)	**8.19** (2.29)	**5.26** (3.57)	**4.20** (4.47)
BCSSTK30	**3.44** (0.35)	**1.92** (0.62)	**1.24** (0.96)	**0.92** (1.29)	**0.90** (1.32)	**1.19** (1.00)
BCSSTK32	**4.08** (0.29)	**2.54** (0.46)	**1.19** (0.99)	**0.85** (1.37)	**0.68** (1.71)	**1.53** (0.77)
M14B	-	-	**51.11** (1.20)	**24.37** (2.51)	**13.54** (4.51)	**9.94** (6.15)
OCEAN	**48.05** (0.27)	**25.22** (0.52)	**13.46** (0.97)	**7.98** (1.63)	**4.78** (2.72)	**3.65** (3.56)
BBMAT	**33.71** (0.37)	**16.30** (0.77)	**9.10** (1.37)	**5.56** (2.25)	**3.51** (3.56)	**3.39** (3.69)
TOOTH	**20.84** (0.30)	**11.67** (0.54)	**6.18** (1.01)	**4.01** (1.56)	**3.03** (2.07)	**2.75** (2.28)
SMDOOR	**32.47** (0.47)	**17.27** (0.89)	**9.86** (1.55)	**5.73** (2.67)	**4.10** (3.73)	**3.54** (4.33)
OILPAN	**7.71** (0.39)	**4.12** (0.73)	**2.39** (1.25)	**1.59** (1.88)	**1.34** (2.23)	**1.59** (1.88)
BMWCRA_1	-	-	**31.54** (1.81)	**17.29** (3.30)	**9.40** (6.07)	**7.21** (8.03)
INVEXTR1	**11.43** (0.33)	**6.11** (0.62)	**3.30** (1.14)	**2.25** (1.67)	**1.87** (2.02)	**1.65** (2.29)
MIXTANK	**16.81** (0.44)	**9.86** (0.74)	**5.46** (1.34)	**3.61** (2.03)	**3.04** (2.41)	**3.09** (2.37)
MT1	-	**24.28** (0.87)	**13.20** (1.60)	**7.68** (2.75)	**5.35** (3.94)	**4.65** (4.54)
QUER	**8.07** (0.41)	**4.31** (0.76)	**2.48** (1.32)	**1.76** (1.86)	**1.57** (2.09)	**1.47** (2.23)
SHIP001	**21.03** (0.43)	**10.72** (0.84)	**5.68** (1.59)	**4.03** (2.24)	**2.31** (3.91)	**2.14** (4.23)
SHIP003	-	-	**45.58** (1.76)	**25.84** (3.10)	**16.05** (4.99)	**12.94** (6.19)
SHIPSEC5	-	-	**38.74** (1.80)	**22.37** (3.11)	**14.49** (4.80)	**11.65** (5.97)
SHIPSEC8	-	**40.28** (0.91)	**22.69** (1.62)	**13.22** (2.79)	**8.69** (4.24)	**8.04** (4.59)
THREAD	**102.14** (0.38)	**41.97** (0.93)	**23.30** (1.67)	**14.01** (2.77)	**12.07** (3.22)	**9.26** (4.20)
X104	**37.80** (0.45)	**20.43** (0.84)	**11.36** (1.51)	**6.97** (2.46)	**5.96** (2.87)	**5.57** (3.07)
BMW3_2	-	**34.65** (0.87)	**18.70** (1.61)	**10.78** (2.79)	**6.98** (4.31)	**5.91** (5.09)
CRANKSG1	-	**31.55** (0.95)	**17.27** (1.74)	**10.27** (2.93)	**6.44** (4.67)	**5.06** (5.95)
CRANKSG2	-	**47.58** (0.97)	**25.87** (1.78)	**14.77** (3.12)	**8.48** (5.43)	**6.34** (7.26)
COLOGB15	**3.68** (0.41)	**1.97** (0.76)	**1.26** (1.18)	**0.94** (1.60)	**0.85** (1.75)	**0.95** (1.57)
COLOGB30	**9.83** (0.44)	**5.23** (0.83)	**3.02** (1.44)	**1.99** (2.19)	**1.62** (2.69)	**1.67** (2.60)
COLOGB75	**37.97** (0.40)	**19.43** (0.79)	**9.22** (1.66)	**5.32** (2.88)	**3.95** (3.87)	**3.41** (4.49)

Tab. C.2: Temps d'exécution (vitesse en Gflops) de la factorisation parallèle pour une distribution 1D pour des cas irréguliers.

Annexe D

Diagrammes de Gantt et de communication comparant la simulation utilisée par le module BLEND et l'exécution réelle du module SOPALIN

Nous comparons ici les traces d'exécution du simulateur de l'étape de repartitionnement et de distribution avec les traces de l'exécution réelle de la factorisation numérique parallèle. On voit que l'ordonnancement prédit conduit à une exécution parallèle efficace qui maximise raisonnablement l'activité des processeurs et la localité des communications. Les traces prédites et réelles sont très semblables (voir figures D.1, D.2, D.3).

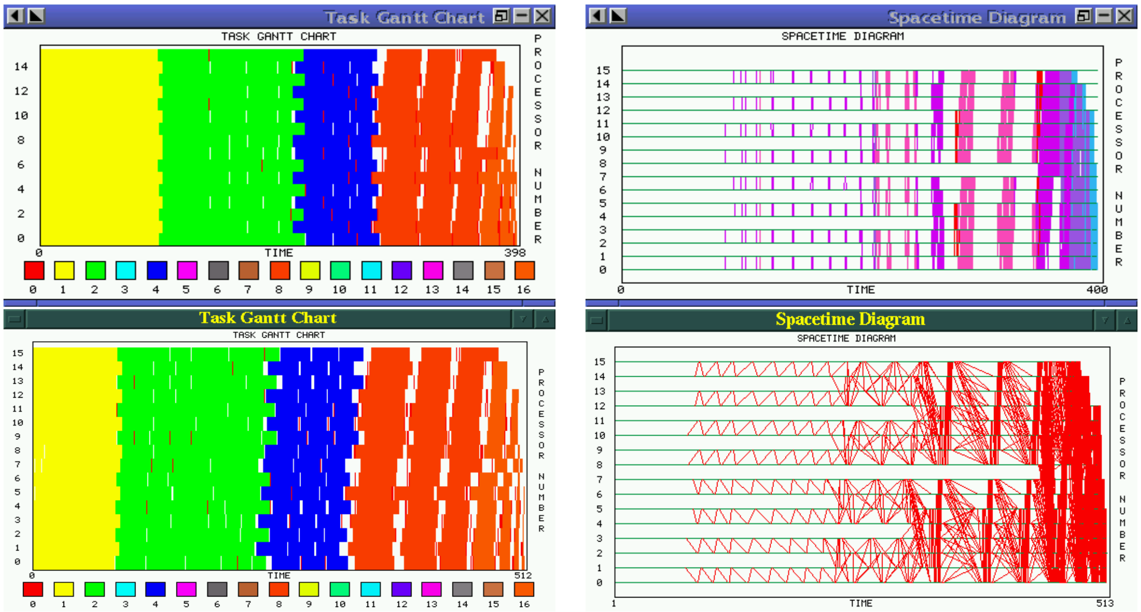

FIG. D.1: CUBE39 : diagramme de Gantt et de communication pour les traces prédites (en haut) et réelles (en bas).

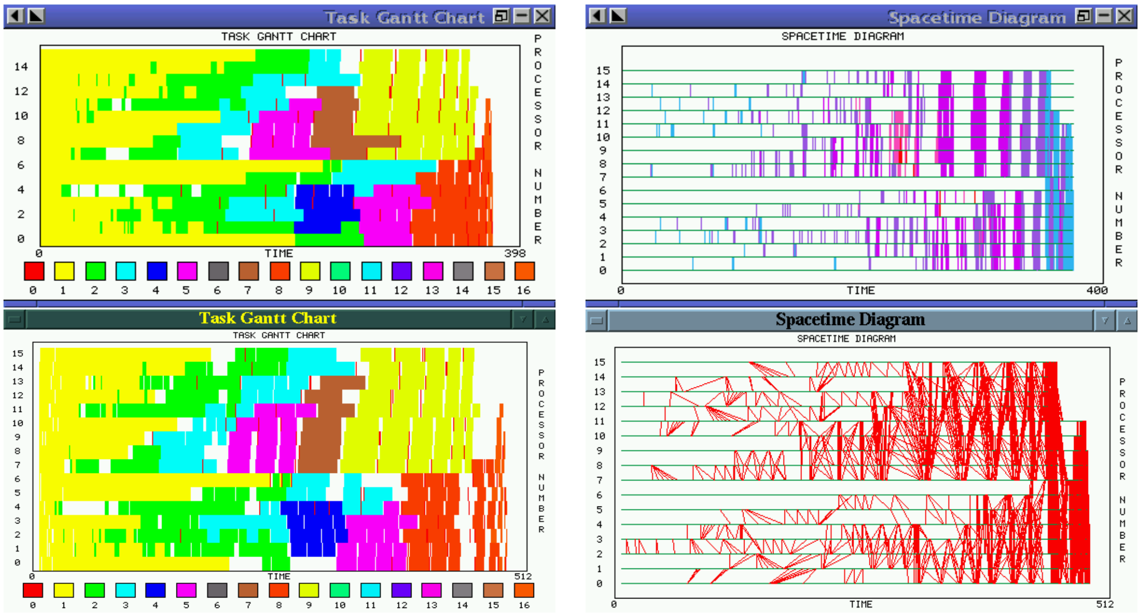

FIG. D.2: BMW3_2 : diagramme de Gantt et de communication pour les traces prédites (en haut) et réelles (en bas).

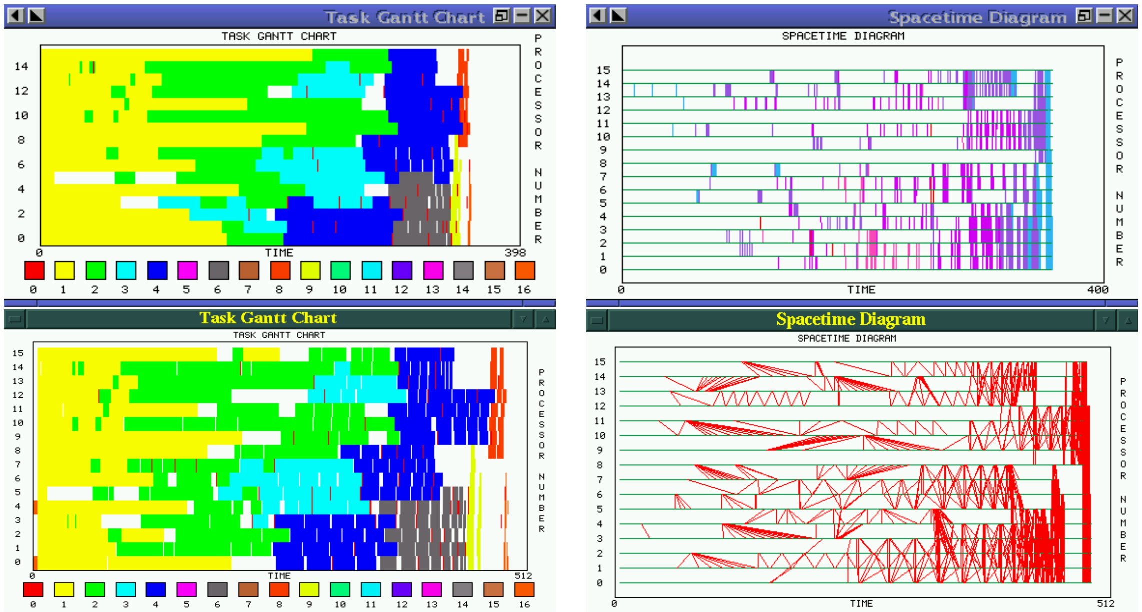

FIG. D.3: CRANKSG1 : diagramme de Gantt et de communication pour les traces prédites (en haut) et réelles (en bas).

Annexe E

Temps pour la descente-remontée parallèle pour une distribution bloc 1D

L'étape de résolution est composée de 2 phases ; la phase de *descente* qui suit exactement le même schéma d'ordonnancement des calculs que celui de la factorisation, et la phase de *remontée* que nous avons implémentée pour suivre l'ordonnancement inverse de la factorisation. En toute rigueur, on pourrait utiliser notre algorithme de distribution adapté aux calculs et aux contraintes de la résolution (via la stratégie utilisée dans le module BLEND), mais cela nécessiterait en général une redistribution (coûteuse) après la factorisation. Nous choisissons donc de conserver la même distribution que pour la factorisation.

Les table E.1 et E.2 présentent les performances de la résolution parallèle par blocs pour notre distribution 1D (le **temps** est donné en secondes et la puissance en gigaflops). Il s'agit des temps obtenus pour réaliser une étape de descente-remontée pour un seul vecteur second membre ; on utilise donc des routines de calcul de type BLAS 2.

Nom	Nombre de processeurs				
	2	4	8	16	32
GRID511	**0.93** (0.05)	**0.49** (0.10)	**0.27** (0.18)	**0.22** (0.22)	**0.19** (0.26)
GRID767	-	**1.04** (0.12)	**0.60** (0.20)	**0.36** (0.33)	**0.27** (0.45)
GRID1023	-	-	**1.07** (0.21)	**0.72** (0.32)	**0.50** (0.45)
CUBE31	**0.33** (0.10)	**0.24** (0.14)	**0.17** (0.19)	**0.12** (0.28)	**0.16** (0.20)
CUBE39	**0.92** (0.10)	**0.58** (0.15)	**0.34** (0.26)	**0.22** (0.40)	**0.23** (0.39)
CUBE47	-	**1.11** (0.17)	**0.64** (0.30)	**0.42** (0.46)	**0.40** (0.49)

TAB. E.1: Temps d'exécution (vitesse en Gflops) de la résolution parallèle pour une distribution 1D pour des cas réguliers.

Nom	Nombre de processeurs				
	2	4	8	16	32
144	-	**1.78** (0.11)	**0.94** (0.20)	**0.57** (0.33)	**0.46** (0.41)
598A	**1.82** (0.06)	**1.03** (0.10)	**0.65** (0.16)	**0.39** (0.26)	**0.27** (0.39)
BCSSTK30	**0.19** (0.09)	**0.24** (0.07)	**0.11** (0.16)	**0.06** (0.31)	**0.05** (0.36)
BCSSTK32	**1.22** (0.02)	**0.16** (0.13)	**0.11** (0.18)	**0.08** (0.28)	**0.07** (0.31)
M14B	-	-	**1.43** (0.17)	**0.82** (0.30)	**0.59** (0.42)
OCEAN	-	**1.51** (0.05)	**0.75** (0.11)	**0.41** (0.20)	**0.35** (0.23)
BBMAT	**0.64** (0.11)	**0.37** (0.19)	**0.24** (0.28)	**0.18** (0.38)	**0.14** (0.47)
TOOTH	**1.34** (0.03)	**0.71** (0.06)	**0.46** (0.09)	**0.27** (0.15)	**0.29** (0.14)
SMDOOR	**1.03** (0.10)	**0.61** (0.17)	**0.41** (0.25)	**0.25** (0.40)	**0.21** (0.50)
OILPAN	**0.43** (0.08)	**0.25** (0.14)	**0.16** (0.22)	**0.13** (0.28)	**0.09** (0.38)
BMWCRA_1	-	-	**1.28** (0.21)	**0.42** (0.63)	**0.49** (0.54)
INVEXTR1	**0.46** (0.06)	**0.33** (0.09)	**0.19** (0.15)	**0.12** (0.24)	**0.12** (0.25)
MIXTANK	**0.44** (0.08)	**0.29** (0.13)	**0.18** (0.20)	**0.16** (0.24)	**0.31** (0.12)
MT1	**1.00** (0.12)	**0.56** (0.22)	**0.34** (0.36)	**0.25** (0.51)	**0.23** (0.53)
QUER	**0.39** (0.09)	**0.23** (0.16)	**0.15** (0.24)	**0.10** (0.37)	**0.08** (0.44)
SHIP001	**0.45** (0.13)	**0.26** (0.22)	**0.22** (0.26)	**0.12** (0.46)	**0.10** (0.55)
SHIP003	-	-	**0.87** (0.27)	**0.48** (0.49)	**0.43** (0.55)
SHIPSEC5	-	-	**0.80** (0.28)	**0.48** (0.48)	**0.45** (0.50)
SHIPSEC8	-	**0.88** (0.16)	**0.51** (0.28)	**0.38** (0.38)	**0.42** (0.34)
THREAD	-	**0.59** (0.16)	**0.39** (0.25)	**0.27** (0.35)	**0.26** (0.37)
X104	**1.28** (0.08)	**0.60** (0.18)	**0.36** (0.29)	**0.26** (0.41)	**0.24** (0.45)
BMW3_2	-	**4.55** (0.04)	**0.72** (0.24)	**0.47** (0.38)	**0.38** (0.47)
CRANKSG1	-	**0.60** (0.21)	**0.39** (0.32)	**0.27** (0.46)	**0.21** (0.61)
CRANKSG2	-	**0.78** (0.21)	**0.50** (0.34)	**0.31** (0.54)	**0.27** (0.62)
COLOGB15	**0.12** (0.11)	**0.10** (0.14)	**0.13** (0.11)	**0.05** (0.29)	**0.05** (0.29)
COLOGB30	**0.25** (0.12)	**0.18** (0.18)	**0.12** (0.26)	**0.07** (0.44)	**0.09** (0.34)
COLOGB75	**0.69** (0.13)	**0.42** (0.21)	**0.26** (0.35)	**0.17** (0.52)	**0.13** (0.69)

TAB. E.2: Temps d'exécution (vitesse en Gflops) de la résolution parallèle pour une distribution 1D pour des cas irréguliers.

Annexe F

Temps de factorisation parallèle pour une distribution bloc mixte 2D/1D

Nous présentons dans cette annexe les performances obtenues par notre solveur avec notre distribution mixte 2D/1D pour les jeux de tests présentés dans la table A.1).

A chaque fois, on a pris le meilleur temps quand on fait varier le seuil dans l'arbre d'élimination pour le changement entre distributions 2D et 1D (le **temps** est donné en secondes et la puissance en gigaflops). On note, comme on pouvait s'y attendre, une meilleure scalabilité qu'à l'annexe C.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
GRID511	**8.25** (0.31)	**4.28** (0.60)	**2.46** (1.04)	**1.56** (1.65)	**1.01** (2.55)	**0.79** (3.26)
GRID767	-	**12.29** (0.71)	**6.63** (1.32)	**4.10** (2.13)	**2.66** (3.29)	**2.01** (4.36)
GRID1023	-	-	**14.57** (1.43)	**8.61** (2.42)	**5.66** (3.68)	**3.75** (5.56)
CUBE31	**11.81** (0.47)	**6.52** (0.85)	**3.89** (1.42)	**2.77** (2.00)	**2.17** (2.55)	**1.88** (2.94)
CUBE39	**43.58** (0.51)	**24.20** (0.93)	**13.58** (1.65)	**8.45** (2.65)	**6.50** (3.45)	**5.03** (4.46)
CUBE47	-	**73.31** (0.95)	**38.23** (1.82)	**23.88** (2.92)	**16.31** (4.27)	**12.45** (5.59)

TAB. F.1: Temps d'exécution (vitesse en Gflops) de la factorisation parallèle pour une distribution 2D/1D pour des cas réguliers.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
144	-	-	**36.06** (1.57)	**21.33** (2.65)	**15.40** (3.67)	**11.83** (4.78)
598A	**78.39** (0.24)	**26.34** (0.71)	**14.40** (1.30)	**8.56** (2.19)	**5.53** (3.39)	**4.45** (4.22)
BCSSTK30	**3.49** (0.34)	**2.00** (0.60)	**1.13** (1.05)	**0.88** (1.36)	**0.77** (1.55)	**0.76** (1.55)
BCSSTK32	**4.03** (0.29)	**2.22** (0.52)	**1.37** (0.85)	**0.98** (1.19)	**0.78** (1.49)	**0.71** (1.63)
M14B	-	-	**46.95** (1.30)	**22.33** (2.74)	**14.06** (4.35)	**9.79** (6.25)
OCEAN	**48.37** (0.27)	**18.39** (0.71)	**9.85** (1.32)	**5.83** (2.23)	**3.82** (3.41)	**3.31** (3.93)
BBMAT	**28.17** (0.44)	**14.73** (0.85)	**8.16** (1.53)	**5.05** (2.47)	**3.64** (3.44)	**2.75** (4.55)
TOOTH	**20.01** (0.31)	**10.70** (0.59)	**6.24** (1.00)	**4.47** (1.40)	**2.93** (2.14)	**2.70** (2.32)
SMDOOR	**29.54** (0.52)	**15.52** (0.99)	**8.86** (1.73)	**5.25** (2.91)	**3.96** (3.87)	**2.91** (5.25)
OILPAN	**7.28** (0.41)	**3.81** (0.78)	**2.15** (1.39)	**1.39** (2.14)	**1.00** (3.00)	**0.87** (3.42)
BMWCRA_1	-	-	**30.97** (1.84)	**17.28** (3.30)	**9.89** (5.76)	**6.94** (8.21)
INVEXTR1	**11.59** (0.32)	**6.31** (0.60)	**3.47** (1.09)	**2.57** (1.47)	**1.78** (2.12)	**1.48** (2.55)
MIXTANK	**17.32** (0.42)	**10.02** (0.73)	**5.91** (1.24)	**3.90** (1.87)	**3.23** (2.27)	**3.17** (2.31)
MT1	**37.92** (0.56)	**20.35** (1.04)	**11.29** (1.87)	**6.65** (3.17)	**4.33** (4.87)	**3.51** (6.01)
QUER	**8.35** (0.39)	**4.46** (0.74)	**2.57** (1.28)	**1.67** (1.96)	**1.16** (2.83)	**0.93** (3.53)
SHIP001	**20.98** (0.43)	**10.91** (0.83)	**6.07** (1.49)	**3.63** (2.49)	**2.43** (3.72)	**1.96** (4.60)
SHIP003	-	**109.78** (0.73)	**45.83** (1.75)	**25.75** (3.11)	**16.60** (4.83)	**12.03** (6.66)
SHIPSEC5	-	**79.12** (0.88)	**35.68** (1.95)	**20.51** (3.39)	**13.99** (4.97)	**11.58** (6.00)
SHIPSEC8	-	**43.40** (0.85)	**24.83** (1.48)	**15.40** (2.39)	**11.39** (3.23)	**9.52** (3.87)
THREAD	**78.04** (0.50)	**41.14** (0.94)	**22.85** (1.70)	**13.50** (2.88)	**10.42** (3.73)	**6.70** (5.80)
X104	**31.53** (0.54)	**19.69** (0.87)	**9.98** (1.72)	**7.49** (2.29)	**5.09** (3.37)	**3.85** (4.44)
BMW3_2	-	**34.02** (0.88)	**18.27** (1.65)	**10.90** (2.76)	**7.69** (3.91)	**6.07** (4.96)
CRANKSG1	-	**33.54** (0.90)	**18.73** (1.61)	**10.54** (2.85)	**6.74** (4.46)	**5.82** (5.17)
CRANKSG2	-	**47.99** (0.96)	**25.33** (1.82)	**14.15** (3.25)	**8.92** (5.16)	**6.20** (7.42)

Tab. F.2: Temps d'exécution (vitesse en Gflops) de la factorisation parallèle pour une distribution 2D/1D pour des cas irréguliers.

Annexe G

Comparaison de performances avec le logiciel PSPASES

Dans cette annexe, nous présentons une comparaison des performances obtenues entre notre chaîne logicielle et la bibliothèque PSPASES [1] version 1.0.3 basée sur une approche multifrontale. Nous utilisons la version 3.4 de Scotch ; PSPASES est configuré pour utiliser la renumérotation de METIS 4.0. La taille critique des blocs pour les BLAS3 est fixée à 64 et la bibliothèque IBM ESSL est utilisée dans les deux cas.

Les jeux de tests utilisés sont ceux pour lesquels on disposait d'une description au format RSA, le format commun accepté par les deux chaînes logicielles. Les mesures données dans la table G.1 ont été obtenues à l'aide d'une factorisation symbolique *scalaire* pour une renumérotation sans amalgamation dans la partie degré minimum : NNZ_{A} est le nombre de termes extra-diagonaux dans la partie triangulaire inférieure de la matrice A, NNZ_{L} est le nombre de termes extra-diagonaux dans la matrice factorisée L et OPC est le nombre d'opérations à réaliser.

Une remarque importante pour aider à la comparaison est que PSPASES réalise une factorisation de type Cholesky (LL^t), intrinsèquement plus efficace au niveau des effets BLAS et des effets de cache que la factorisation LDL^t que nous utilisons. Par exemple, pour une matrice pleine 1024x1024, sur un nœud Power2SC, la routine ESSL effectuant la factorisation LL^t prend 1.07s, alors que la routine ESSL effectuant la factorisation LDL^t prend 1.27s.

[1] Parallel SPArse Symmetric dirEct Solver.

Nom	Taille	NNZ_A	$NNZ_{L\,(\text{Scotch})}$	OPC(Scotch)	$NNZ_{L\,(\text{METIS})}$	OPC(METIS)
SMDOOR	162610	3873534	2.542e+07	1.531e+10	2.404e+07	1.237e+10
BMWCRA_1	148770	5247616	6.597e+07	5.702e+10	6.981e+07	6.124e+10
MT1	97578	4827996	3.115e+07	2.109e+10	3.455e+07	2.269e+10
OILPAN	73752	1761718	8.912e+06	2.985e+09	9.065e+06	2.751e+09
QUER	59122	1403689	9.119e+06	3.281e+09	9.586e+06	3.448e+09
SHIP001	34920	2304655	1.428e+07	9.034e+09	1.481e+07	9.462e+09
SHIP003	121728	3982153	5.873e+07	8.008e+10	5.910e+07	7.587e+10
SHIPSEC5	179860	4966618	5.650e+07	6.952e+10	5.256e+07	5.509e+10
THREAD	29736	2220156	2.404e+07	3.884e+10	2.430e+07	3.583e+10
X104	108384	5029620	2.634e+07	1.713e+10	2.728e+07	1.412e+10

Tab. G.1: Description des cas tests de comparaison.

La table G.2 présente les performances obtenues pour la factorisation parallèle (le temps est donné en secondes et la puissance en gigaflops). Pour chaque matrice, la première ligne donne le temps du solveur 2D de la chaîne EMILIO, et la seconde celui de PSPASES. Les tirets correspondent à des exécutions faussées par des phénomènes de "swapping" ou d'allocation mémoire impossible.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
B5TUER	**29.54** (0.52)	**15.52** (0.99)	**8.86** (1.73)	**5.25** (2.91)	**3.96** (3.87)	**2.91** (5.25)
	-	-	14.04 (0.88)	7.32 (1.69)	3.91 (3.16)	2.58 (4.79)
BMWCRA1	-	-	**30.97** (1.84)	**17.28** (3.30)	**9.89** (5.76)	**6.94** (8.21)
	-	-	-	-	24.87 (2.49)	13.48 (4.61)
MT1	**37.92** (0.56)	**20.35** (1.04)	**11.29** (1.87)	**6.65** (3.17)	**4.33** (4.87)	**3.51** (6.01)
	-	-	-	10.71 (2.12)	5.70 (3.98)	3.59 (6.32)
OILPAN	**7.28** (0.41)	**3.81** (0.78)	**2.15** (1.39)	**1.39** (2.14)	**1.00** (3.00)	**0.87** (3.42)
	-	5.23 (0.53)	2.79 (0.99)	1.73 (1.59)	1.25 (2.20)	0.93 (2.96)
QUER	**8.35** (0.39)	**4.46** (0.74)	**2.57** (1.28)	**1.67** (1.96)	**1.16** (2.83)	**0.93** (3.53)
	23.80 (0.14)	13.11 (0.26)	3.22 (1.07)	2.01 (1.72)	1.30 (2.65)	0.96 (3.59)
SHIP001	**20.98** (0.43)	**10.91** (0.83)	**6.07** (1.49)	**3.63** (2.49)	**2.43** (3.72)	**1.96** (4.60)
	-	15.32 (0.62)	8.11 (1.17)	4.48 (2.11)	2.98 (3.17)	2.14 (4.42)
SHIP003	-	**109.78** (0.73)	**45.83** (1.75)	**25.75** (3.11)	**16.60** (4.83)	**12.03** (6.66)
	-	-	-	-	21.28 (3.57)	14.08 (5.39)
SHIPSEC5	-	**79.12** (0.88)	**35.68** (1.95)	**20.51** (3.39)	**13.99** (4.97)	**11.58** (6.00)
	-	-	-	-	21.80 (2.52)	11.81 (4.66)
THREAD	**78.04** (0.50)	**41.14** (0.94)	**22.85** (1.70)	**13.50** (2.88)	**10.42** (3.73)	**6.70** (5.80)
	-	-	-	21.02 (1.70)	11.24 (3.19)	6.41 (5.59)
X104	**31.53** (0.54)	**19.69** (0.87)	**9.98** (1.72)	**7.49** (2.29)	**5.09** (3.37)	**3.85** (4.44)
	-	-	-	8.32 (1.70)	4.92 (2.87)	3.18 (4.44)

Tab. G.2: Comparaison des performances avec PSPASES.

Dans la quasi-totalité des cas où il n'y a pas de problème de granularité, la comparaison est favorable à notre solveur 2D au moins jusqu'à 64 processeurs, et ce même si la scalabilité de PSPASES semble à priori meilleure. Les travaux en cours sur une distribution mixte 2D/1D ont en particulier pour objectif de traiter ce problème.

Annexe H

Liste des publications de l'auteur

Revues internationales avec comité de lecture :

[1] D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. *accepté dans Numerical Algorithms, Baltzer Science Publisher*, 2000, à paraître.

Congrès internationaux avec comité de sélection et avec actes :

[2] David Goudin and Jean Roman. A scalable parallel assembly for irregular meshes based on a block distribution for a parallel block direct solver. In *Proceedings of PARA'2000*, Lecture Notes in Computer Science. Springer Verlag, june 2000, Bergen, à paraître.

Congrès internationaux sur invitation :

[3] D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman et J.-J. Pesqué. *Parallel Sparse Linear Algebra and Application to Structural Mechanics.* In *SP-World'99, Montpellier, France*, mai 1999.

[4] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. In *European ACTC Workshop 2000, Paris, France*, mai 2000.

[5] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Description of the emilio software processing chain and application to structural mechanics. In *PMAA2000, Lausanne, Suisse*, août 2000.

Congrès nationaux avec comité de sélection et avec actes :

[6] D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman et J.-J. Pesqué. *Algèbre Linéaire Creuse Parallèle pour les Méthodes Directes : Application à la Parallélisation d'un Code de Mécanique des Structures.* Journées sur l'Algèbre Linéaire Creuse et ses Applications Industrielles, IRISA, Rennes, mars 1999.

Rapports de contrats :

[7] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1998. Premier Rapport Semestriel.

[8] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1999. Rapport Final de la Première Partie.

[9] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1999. Rapport Semestriel de la Deuxième Partie.

Bibliographie

[1] P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. and Appl.*, pages 886–905, 1996.

[2] P. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. Supercomputer Applics*, 7 :64–82, 1993.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. Technical Report RT/APO/99/2, ENSEEIHT-IRIT, 1999. To appear in SIAM Journal on Matrix Analysis and Applications.

[4] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. In *Comput. Methods Appl. Mech. Eng.*, pages 501–520. 2000.

[5] A. Ashcraft. The fan-both family of column-based distributed cholesky factorization algorithms. In Alan George, John R. Gilbert, and Joseph W.-H. Liu editors, editors, *Graph Theory and Sparse Matrix Computations*. Springer-Verlag, New York, NY, 1993.

[6] A. Ashcraft, S. C. Eisenstat, and J. W.-H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM Journal on Scientific and Statistical Computing*, 11 :593–599, 1990.

[7] A. Ashcraft, S. C. Eisenstat, J. W.-H. Liu, and A. H. Sherman. A comparison of three column based distributed sparse factorization schemes. Technical Report YALEU/DCS/RR-810, Yale University, New Haven, 1990.

[8] C. Ashcraft. The fan-both family of column-based distributed Cholesky factorization algorithms. *Graph Theory and Sparse Matrix Computation, IMA*, 56 :159–190, 1993.

[9] C. Ashcraft, S. C. Eisenstat, and J. W.-H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM J. Sci. Stat. Comput.*, 11(3) :593–599, 1990.

[10] C. Ashcraft, S. C. Eisenstat, J. W.-H. Liu, B. W. Peyton, and A. Sherman. A compute-ahead implementation of the fan-in sparse distributed factorization

scheme. Technical report, Departement of Computer Science, York University, September 1994.

[11] C. Ashcraft, S. C. Eisenstat, J. W.-H. Liu, and A. Sherman. A comparison of three column based distributed sparse factorization schemes. In *Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, 1991.

[12] C. Ashcraft and R. G. Grimes. SPOOLES : An objected oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 1999.

[13] M. Benzi, J. K. Cullum, and M. Tuma. Robust Approximate Inverse Preconditionning for the Conjugate Gradient Method. Technical Report LA-UR-99-2899, Los Alamos National Laboratory, June 1999. To appear in SIAM J. Sci. Comput.

[14] V. E. Bulgakov and G. Kuhn. High performance multilevel iterative aggregation solver for large finite element structural analysis problems. *I.J.N.M.E.*, 38 :3529–3544, 1995.

[15] P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, 55 :463–476, 1989.

[16] J. M. Conroy, S. G. Kratzer, and R. F. Lucas. Multifrontal sparse solvers in message passing and data parallel environments - a comparitive study. In *Proceedings of PARCO*, 1993.

[17] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and non linear equations.* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[18] J. K. Dickinson and P. A. Forsyth. Preconditioned conjugate gradient methods for 3d linear elasticity. *I.J.N.M.E.*, 37 :2211–2234, 1994.

[19] M. C. Dracopoulos and M. A. Crisfield. Coarse/fine mesh preconditioners for the iterative solution of finite element problems. *I.J.N.M.E.*, 38 :3297–3313, 1995.

[20] I. S. Duff. Sparse numerical linear algebra : direct methods and preconditioning. Technical Report TR/PA/96/22, CERFACS, 1996.

[21] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information and Support Services and Report TR/PA/97/36 from CERFACS, Toulouse.

[22] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9 :302–325, 1983.

[23] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, 5(3) :633–641, 1984.

[24] L. Facq and J. Roman. Algèbre linéaire creuse : distribution par bloc pour une factorisation parallèle de cholesky. In G. Authié, J. M. Garcia, A. Ferreira, J. L. Roch, G. Villard, J. Roman, C. Roucairol, and B. Virot editors, editors, *Parallélisme et applications irrégulières*, pages 135–147. Hermès, 1995.

[25] C. Farath, K. Pierson, and M. Lesornne. The second generation of feti methods and their application to the parallel solution of large-scale linear and geometrically non linear structural analysis problems. *Computer Methods in Applied Mechanics and Engineering, C.M.A.M.E.*, 184, 2000.

[26] Fletcher. *Practical methods of optimization.* J. Wiley and Sons, 1987.

[27] G. A. Geist and E. Ng. Task scheduling for parallel sparse Cholesky factorization. *Internat. J. Parallel Programming*, 18(4) :291–314, 1989.

[28] A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[29] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1998. Premier Rapport Semestriel.

[30] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1999. Rapport Final de la Première Partie.

[31] D. Goudin. Mise en oeuvre d'une bibliothèque d'outils pour la résolution par méthode directe de grands systèmes linéaires creux symétriques définis positifs sur machine parallèle. Technical report, C.E.A. / C.E.S.T.A, 1999. Rapport Semestriel de la Deuxième Partie.

[32] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. In *SP-World'99, Montpellier*, 1999.

[33] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Description of the emilio software processing chain and application to structural mechanics. In *PMAA2000, Lausanne*, 2000.

[34] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. In *European ACTC Workshop 2000, Paris*, 2000.

[35] D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. *accepté dans Numerical Algorithms, Baltzer Science Publisher*, 2000, à paraître.

[36] D. Goudin, F. Pellegrini, and J. Roman. Pré-étude de la parallélisation du code ossau de mécanique des structures. Technical report, C.E.A. / C.E.S.T.A, 1997. Étude commandée par le C.E.A. / C.E.S.T.A.

[37] D. Goudin and J. Roman. A scalable parallel assembly for irregular meshes based on a block distribution for a parallel block direct solver. In *Proceedings of PARA'2000*, Lecture Notes in Computer Science. Springer Verlag, june 2000, Bergen, à paraître.

[38] E. Graham and P. A. Forsyth. Preconditioning methods for very ill conditioned 3d linear elasticity problems. *I.J.N.M.E.*, 44 :77–99, 1999.

[39] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. IEEE Trans. on Parallel and Distributed Systems, 8(5) :502–520, May 1997.

[40] A. Gupta and V. Kumar. A scalable parallel algorithm for sparse matrix factorization. Technical Report 94-19, Department of Computer Science, University of Minnesota, Minneapolis, 1994.

[41] A. Gupta and E. Rothberg. An evaluation of left-looking, right-looking, and multifrontal approaches to sparse cholesky factorization on hierarchical-memory machines. *Int. J. High Speed Comput.*, 5, 1993.

[42] B. Hendrickson and R. Leland. The *chaco* user's guide. Technical Report SAND93–2339, Sandia National Laboratories, November 1993.

[43] P. Henon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *Proceedings of EuroPAR'99*, number 1685 in Lecture Notes in Computer Science, pages 1059–1067. Springer Verlag, 1999.

[44] P. Henon, P. Ramet, and J. Roman. PaStiX : A Parallel Sparse Direct Solver Based on a Static Scheduling for Mixed 1D/2D Block Distributions. In *Proccedings of Irregular'2000*, number 1800 in Lecture Notes in Computer Science, pages 519–525. Springer Verlag, May 2000.

[45] M. Joshi, G. Karypis, V. Kumar, A. Gupta, and Gustavson F. PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems. Technical report, University of Minnesota and IBM Thomas J. Watson Research Center, May 1999.

[46] G. Karypis and V. Kumar. MeTiS – *Unstructured Graph Partitioning and Sparse Matrix Ordering System – Version 2.0.* University of Minnesota, June 1995.

[47] P. Laborde, B. Toson, and J.-J. Pesqué. On the consistent tangent operator algorithm for thermo-pastic problems. *Comp. Methods Appl. Mech. Eng.*, 146 :215–230, 1997.

[48] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and Krogh F. T. Algorithm 539. basic linear algebra subprograms for fortran usage. In *ACM Transactions on Mathematical Software*, volume 5, pages 324–325. 1979.

[49] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and Krogh F. T. Basic linear algebra subprograms for fortran usage. In *ACM Transactions on Mathematical Software*, volume 5, pages 308–323. 1979.

[50] F. J. Lingen. A generalized conjugate residual method for the solution of non symetric systems of equations with multiple right hand sides. *I.J.N.M.E.*, 43 :641–656, 1999.

[51] J. W.-H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11 :134–172, 1990.

[52] T. A. Manteuffel. The shifted incomplete cholesky factorization. Technical report, Sandia Laboratories Livermore, 1978.

[53] Minoux. *Programmation mathématique, théorie et algorithmes.* Dunod, 1983.

[54] J. A. Mitchell and J. N. Reddy. A multilevel hierarchical preconditioner for thin elastic solids. *I.J.N.M.E.*, 43 :1383–1400, 1998.

[55] E. Ng and B. W. Peyton. Block sparse cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14 :1034–1056, 1993.

[56] F. Pellegrini and J. Roman. Sparse matrix ordering with SCOTCH. In *Proceedings of HPCN'97*, number 1225 in Lecture Notes in Computer Science, pages 370–378. Springer Verlag, April 1997.

[57] F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. In *Proceedings of Irregular'99*, number 1586 in Lecture Notes in Computer Science, pages 986–995. Springer Verlag, April 1999. Extended paper appeared in Concurrency : Practice and Experience, 12 :69-84, 2000.

[58] A. Pothen and C. Sun. A mapping algorithm for parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 14(5) :1253–1257, September 1993.

[59] J. Roman. Partitionnement algorithmique des données pour la factorisation de cholesky par bloc de grands systèmes linéaires creux sur des calculateurs mimd. *Lettre du transputer et des calculateurs parallèles*, 6(24) :115–120, 1994.

[60] E. Rothberg. Performance of panel and block approaches to sparse Cholesky factorization on the iPSC/860 and Paragon multicomputers. *SIAM J. Sci. Comput.*, 17(3) :699–713, May 1996.

[61] E. Rothberg and A. Gupta. An efficient block-oriented approach to parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 15(6) :1413–1439, November 1994.

[62] E. Rothberg and A. Gupta. An efficient block-oriented approach to parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 15(6) :1413–1439, November 1994.

[63] E. Rothberg and R. Schreiber. Improved load distribution in parallel sparse Cholesky factorization. In *Proceedings of Supercomputing'94*, pages 783–792. IEEE, 1994.

[64] Y. Saad. *Iterative Methods For Sparse Linear Systems.* Ed. PWS publishing Compagny, 1996.

[65] R. Schreiber. Scalability of sparse direct solvers. Technical Report TR 92.13, RIACS, NASA Ames Research Center, May 1992.

[66] Y. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *BIT*, 1999. To appear.

[67] M. Suarjana and Law K. A robust incomplete factorization based on value and space constraints. *I.J.N.M.E.*, 38 :1703–1719, 1995.

[68] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *J. Proc. IEEE*, 55 :1801–1809, 1967.

[69] Wilson and Bathe. *Numerical methods and finite element analysis.* Prentice-Hall, 1976.

[70] O.C. Zienkiewicz. *La méthode des Elements Finis.* Ed. Mac Graw-Hill, 1979.